Actas de las I Jornadas sobre Algoritmos Evolutivos y Metaheurísticas

JAEM'07

Editadas por
Enrique Alba
Francisco Chicano
Francisco Herrera
Francisco Luna
Gabriel Luque
Antonio J. Nebro

Zaragoza, 12 y 13 de Septiembre de 2007

Índice General

Una heurística Beam Search para el problema de Equilibrado de Líneas de Montaje

Joaquín Bautista
Cátedra Nissan UPC
Universitat Politècnica de Catalunya
08028 Barcelona
joaquin.bautista@upc.edu

Jordi Pereira
Dept. de Organización de Empresas
Universitat Politècnica de Catalunya
08028 Barcelona
jorge.pereira@upc.edu

Resumen

El problema de equilibrado de líneas de montaje simple (SALBP-1) es una simplificación del problema asociado al reparto de las tareas en que se descompone el ensamblaje de un producto en una línea de montaje, éste ha sido extensamente estudiado en la literatura. El presente trabajo expone un nuevo procedimiento de resolución basado en la metaheurística *Beam Search*. El procedimiento emplea un esquema constructivo específicamente diseñado para el problema y su aplicación ha demostrado ofrecer soluciones de calidad, ya que es capaz de obtener la mejor solución conocida para 266 de las 269 instancias presentes en la literatura del SALBP.

1. Introducción

El problema de equilibrado de líneas de montaje (ALBP) es un problema clásico que ha sido tratado al menos durante los últimos 50 años; sirvan de ejemplo los trabajos de Salveson [16] y Jackson [11]. Simplificadamente, equilibrar una línea de montaje consiste en asignar las tareas en que se divide el montaje de un producto acabado entre las diferentes estaciones que componen la línea en la que se ensambla el producto. Dichas estaciones se disponen en serie y el producto pasa de una estación a la siguiente cuando finaliza el tiempo concedido a la estación para realizar las tareas asignadas a ella. El tiempo concedido a todas las estaciones, por igual, se conoce como tiempo de ciclo y su valor inverso coincide con la tasa de producción de la línea en su conjunto.

El objetivo asociado a estos problemas suele ser la minimización del tiempo muerto en la línea, para ello, se intenta minimizar indirectamente el número de estaciones de trabajo o el tiempo de ciclo o, bajo determinadas condiciones, una combinación de ambos términos. También puede plantearse un problema de factibilidad dado un tiempo de ciclo y un número de estaciones.

Una clasificación propuesta en [4] divide todos los problemas de equilibrado en dos grandes categorías: aquellos que se denominan simples, SALBP, y el resto de casos que se denominan generales, GALBP. Según dicha clasificación los SALBP son aquellos problemas que tienen en cuenta únicamente dos tipos de restricciones en cuanto a la asignación de tareas a las estaciones:

- Restricciones de tipo acumulativo, asociadas al tiempo disponible en cada estación para operar sobre el producto.
- Restricciones de tipo potencial, asociadas a que ciertas tareas deben realizarse tras la conclusión de otras.

Cualquier problema que tenga en cuenta otros elementos, como incompatibilidades entre tareas [1], disposición de estación en líneas U [14], restricciones adicionales sobre el espacio en las estaciones [3] o estaciones paralelas [8], entre muchas otras, quedan incluidas en la categoría GALBP.

Aunque la inmensa mayoría de problemas encontrados en contexto industrial forman parte de la categoría GALBP, la comunidad científica ha dedicado grandes esfuerzos al desarrollo de procedimientos para resolver los casos SALBP, debido, en parte, a que es el modelo que subyace en los casos generales y, por ello, los resultados obtenidos a partir de SALBP pueden aplicarse o servir al menos de base para la resolución de casos generales. Entre los diferentes modelos SALBP, según la función objetivo, el más común es el denominado SALBP-1, el cual consiste en hallar el número mínimo de estaciones, que contengan todas las tareas, tras fijar un valor para el tiempo de ciclo o la tasa de producción de la línea.

En dos trabajos recientes, [3] y [6], basados en algoritmos de colonias de hormigas (ACO), se ha

mostrado que las aproximaciones metaheurísticas resultan competitivas para resolver el problema de equilibrado de líneas de montaje simple.

La propuesta de [6] es en la actualidad la mejor heurística disponible en la literatura. Dicha heurística combina las ideas de la metaheurística *Beam Search*, [15], con los algoritmos de colonias de hormigas, [6], para mejorar los resultados obtenidos por cada uno de ellos individualmente.

En el presente trabajo nos hemos centrado en el estudio el comportamiento de la metaheurística *Beam Search* (BS) de forma independiente a los algoritmos de hormigas. Usando algunas mejoras (asociadas a la mejor explotación de la estructura de las buenas soluciones) en la fase constructiva del algoritmo, nos ha permitido disponer de un procedimiento que supera los resultados obtenidos en los trabajos anteriores empleando tiempos de computación reducidos. Además, con BS se mejoran, por primera vez en la literatura, los resultados obtenidos por el mejor procedimiento exacto conocido.

2. Revisión de la literatura

La comunidad científica ha desarrollado un gran número de propuestas para resolver el problema de equilibrado de líneas de montaje simple, véase un reciente estado del arte en [20], así como trabajos de descripción y resolución de problemas generales, véase [5], o la clasificación presentada en [7].

En resumen, los procedimientos de resolución de problemas de equilibrado pueden dividirse en tres grupos:

(1) Un primer grupo formado por procedimientos constructivos de tipo "*greedy*" que utilizan una regla de prioridad estática o dinámica para asignar las tareas a las diferentes estaciones, véase [21]. Estos procedimientos ofrecen buenas soluciones con unos tiempos de computación muy reducidos.

(2) Un segundo grupo formado por procedimientos de enumeración, basados generalmente en procedimientos de exploración arborescente tipo *Branch and Bound*, o de exploración de grafos como la Programación Dinámica, véase [19]. Entre los diferentes procedimientos que forman parte de este grupo cabe destacar SALOME, [18], un algoritmo *Branch and Bound* que puede considerarse el mejor algoritmo disponible para resolver esta clase de problemas y que es capaz de resolver de forma optima 260 de las 269 instancias utilizadas en la literatura, alcanzando la mejor solución conocida para otra de las instancias.

(3) Un tercero formado por procedimientos metaheurísticos. Aunque sus resultados no son comparables, por lo general, con los métodos del segundo grupo, resultan importantes ya que su aplicabilidad para problemas generales es superior a la de los métodos exactos, que dependen en gran medida de cotas cuya calidad disminuye en los problemas generales. Entre las propuestas más efectivas para este problema aparecen los algoritmos de Búsqueda Tabú [17] y recientemente [12], el algoritmo de hormigas propuesto previamente por los autores, [3], o el algoritmo *Beam-ACO* de [6]. Este último es el que mejores resultados ha obtenido para la colección de instancias de la literatura pudiendo resolver 245 de las 269 instancias que se utilizan normalmente para comparar los algoritmos propuestos.

La calidad de los métodos constructivos, que pueden resolver un gran número de instancias de la literatura sin ayuda de otros procedimientos, la existencia de procedimientos de acotación de alta calidad en bajos tiempos de cálculo [19], y la dificultad para definir vecindarios para el SALBP-1, véase [3] para un análisis de este último punto, son tres de las posibles razones por las cuales los algoritmos de exploración arborescente resultan mejores que los procedimientos de mejora local, los cuales son básicos en el buen comportamiento de muchos procedimientos metaheurísticos. Por ejemplo, el algoritmo presentado en [6] no hace uso de ningún procedimiento de búsqueda local para mejorar las soluciones obtenidas.

Es por ello que muchos de los trabajos desarrollados en este campo se basan en mayor o menor medida en procedimientos de exploración arborescente. Entre ellos destacamos la propuesta encontrada en [9] que emplea un procedimiento conocido como la heurística de Hoffmann, [10]. El procedimiento propuesto se basa en la enumeración de todas las asignaciones posibles en cada estación por separado, escogiendo como mejor asignación aquélla que causa el menor tiempo muerto en la estación en construcción. Este procedimiento puede entenderse como la resolución de múltiples problemas Knapsack con restricciones adicionales asociadas al diseño de

cada estación por separado. Adicionalmente, en [9] se hace uso de cotas y otras propiedades conocidas del problema para mejorar el algoritmo original, obteniendo un procedimiento que es capaz de resolver más de 200 instancias de la referida colección de instancias de la literatura.

El presente trabajo se centra en la aplicación de la metaheurística *Beam Search* al problema. El principio básico de la metaheurística *Beam Search*, [15], consiste en la enumeración parcial de un árbol de exploración. Partiendo de las ideas de la búsqueda *Branch and Bound* con una regla de exploración "primero en anchura", se aplican dos criterios para descartar, de forma heurística, vértices del árbol de exploración con el objetivo de obtener soluciones al problema en tiempos de computación inferiores a la exploración completa del árbol.

El primer criterio es limitar el número de vértices generados (soluciones parciales bajo estudio) a partir de un vértice concreto (solución parcial); la generación se controla a través de un parámetro denominado *beamextensions*. Este parámetro indica el número máximo de soluciones parciales que se desarrollarán partiendo de la solución parcial incumbente.

El segundo criterio, asociado a la exploración "primero en anchura" del árbol, limita el número de soluciones parciales de un nivel del árbol de búsqueda que se continuarán explorando en el siguiente nivel del árbol. Este criterio se controla a través de un parámetro denominado *beamwidth*, que se utilizará para seleccionar el número de "mejores" soluciones parciales a desarrollar, entendiendo por "mejor" algún criterio específico del problema.

El presente trabajo hace uso de esta estructura para proponer un nuevo algoritmo de resolución para el SALBP-1. El resto del trabajo presenta el contenido que sigue. La sección 3 muestra una formulación del problema tratado, así como ciertas propiedades de dicho problema de interés para el desarrollo del presente trabajo, como la reversibilidad de las instancias o algunas de las cotas disponibles en la literatura. En la sección 4 se muestra el esquema del procedimiento desarrollado, prestando especial interés a las dos fases principales del algoritmo asociadas a: (1) la creación de nuevas soluciones parciales, partiendo de aquéllas que están en curso, bajo el control del parámetro *beamextensions*, y (2) la selección de un subconjunto de ellas, bajo el control del parámetro *beamwidth*. Tras la presentación del algoritmo, la sección 5 documenta la experiencia computacional desarrollada para comprobar la validez de la propuesta realizada, mostrando que el algoritmo implementado puede considerarse parte del estado del arte como procedimiento de resolución del problema estudiado. Finalmente la sección 6 se destina a mostrar las conclusiones del presente trabajo y los posibles desarrollos futuros.

3. Formulación del SALBP-1

Formalmente, una instancia SALBP-1 queda definida por un conjunto de tareas elementales V, indivisibles, en que se ha dividido el montaje del producto acabado. Cada una de las tareas tiene una duración determinista y conocida, d_i, (i=1,…,$|V|$), que sin pérdida de generalidad puede considerarse un valor entero. Se dispone de un conjunto ordenado de estaciones S_j, (j=1,…,m) cada una de ellas con un tiempo concedido idéntico e igual al tiempo de ciclo, c. Nótese que una cota del número de estaciones equivale al número de tareas del problema, ya que existe una solución trivial consistente en asignar una tarea a cada estación, esto es $m \leq |V|$. Adicionalmente, algunas tareas presentan relaciones de precedencia entre ellas, por ejemplo es conveniente montar los asientos de un coche antes de instalar las puertas.

La relaciones de precedencia entre tareas suelen representarse mediante un grafo acíclico G(V,A) donde existe un arco (i,k) 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.

El objetivo del problema consiste en determinar una asignación de tareas (qué tareas deben realizarse en cada estación) tal que se minimice el número de estaciones m, esto es:

(1) $\Sigma_{\forall i \in S_j} d_i \leq c$, para todo S_j, (j=1,…,m)

(2) $j \leq l$, para todo $(i,k) \in G(V,A)$, $i \in S_j$ y $k \in S_l$

(3) $i \in S_1 \cup ... \cup S_m$, para todo i, (i=1,…,$|V|$)

(4) $S_j \cap S_l = \emptyset$, $j<l$ (j=1,..,m-1 y l =j+1,..,m)

Las restricciones (1) se refieren al tiempo concedido en cada estación, el conjunto (2) fuerza el cumplimiento de las restricciones de precedencia entre tareas, (3) obligan a que cada tarea se asigne al menos a una estación y (4) obligan a que no se compartan las tareas entre las estaciones .

De la formulación anterior puede verse que el problema equivale a un problema de Bin Packing, BP-1 siguiendo la nomenclatura de Martello y Toth, [13], al que se le añaden las restricciones de precedencia (2), por ello, cualquier procedimiento exacto o de acotación de dicho problema es útil para obtener cotas del problema SALBP-1. En este trabajo se utilizan la cotas simple LB1, [4], consistente en determinar el número mínimo de estaciones necesarias si fuera posible dividir las tareas entre varias estaciones, $LB1=\lceil \Sigma_{\forall i \in V} d_i/c \rceil$.

Además, el problema es reversible: si el sentido de los arcos del grafo de precedencias G(V,A) se invierte, una solución obtenida para esta instancia puede transformarse en una solución para la instancia original: basta cambiar la ordenación de las estaciones obtenidas, $S_j \leftarrow S_{m+1-j}$, para todo S_j, (j=1,…,m). Esta propiedad también se utiliza en el presente trabajo ya que es obvio que las soluciones obtenidas para la instancia directa o inversa pueden ser diferentes.

4. Descripción del algoritmo

El algoritmo implementado parte de las ideas básicas de la metaheurística *Beam Search*. Partiendo de un enfoque *Branch and Bound* con un criterio de búsqueda primero en anchura, se inicia el procedimiento con una solución parcial vacía. De dicha solución se generan un máximo de *beamextensions* soluciones parciales que han sido construidas mediante la incorporación de una nueva estación a la solución parcial y la asignación de todas las tareas que formarán parte de dicha estación. Entre ellas se escoge un máximo de *beamwidth* soluciones parciales y se repite el procedimiento mostrado para cada una de ellas, seleccionado sólo un máximo de *beamwidth* soluciones parciales de un total máximo de *beamwidth·beamextensions* soluciones parciales posibles.

La descripción del algoritmo se ha dividido en dos partes. La primera corresponde al método de construcción de soluciones parciales y que se basa en el algoritmo de Hoffmann, [10], responsable de generar las extensiones, y la segunda corresponde al procedimiento de búsqueda propiamente dicho basado en la metaheurística *Beam Search*, [15].

4.1. Construcción de soluciones

La enumeración se basa en Hoffmann, [10]. Este procedimiento aplica un algoritmo de búsqueda en que consecutivamente se busca la asignación de tareas a la estación en curso retornando una asignación sin tiempo libre en la estación o la mejor solución encontrada en términos de menor tiempo libre. La propuesta original utiliza el procedimiento descrito repetidamente hasta que no quedan más tareas por asignar, en cuyo caso se ha encontrado una solución para la instancia.

El procedimiento de Hoffmann tiene el inconveniente de transformar el problema original en una cadena de problemas de Knapsack con restricciones adicionales, siendo éste un problema difícil de resolver por sí solo. La teórica complejidad del problema, que forma parte de la categoría NP Hard, puede hacer pensar que el procedimiento no es aplicable dentro de una metaheurística, pero los tiempos de computación observados, del orden de milésimas de segundo para las instancias estudiadas, resultan razonables. Para mejorar la efectividad del procedimiento, se han aplicado una serie de modificaciones en la implementación del mismo que se detallan a continuación:

- *Mantenimiento de las estaciones construidas y modificación de la condición de final*. Para poder obtener más de una extensión partiendo de una solución parcial, el algoritmo almacena un número máximo de *beamextensions* soluciones parciales, ordenadas de menor a mayor tiempo ocioso en la estación en construcción. Cuando no es posible construir una asignación diferente o todas las asignaciones almacenadas no contienen tiempos muertos para la estación en construcción, se devuelve dicho conjunto.

- *Reordenación de las tareas*. Es conocido que el tiempo de cálculo necesario durante un procedimiento de enumeración depende en gran medida del orden en que las tareas son analizadas. Para paliar dicho efecto se reordenan las tareas para evitar tiempo de cálculo innecesario. Las tareas se ordenan en una lista según los siguientes criterios:

(1) Una tarea con precedencias siempre aparece posteriormente en la lista a cualquiera de sus tareas predecesoras.

(2) Las tareas están en orden no decreciente a su cota inferior de asignación. Antes de reordenar las tareas se determina la primera estación a la que pueden asignarse calculando el número de estaciones que requieren todas sus tareas

predecesoras más ella misma y se aplica la cota LB1 a ese conjunto de tareas; descartando tareas cuya cota inferior de estación supere al número de orden de la estación en construcción.

(3) En caso de empate en el criterio anterior, aquellas tareas con mayor duración aparecen antes en la lista.

(4) Entre aquellas tareas con igual cota inferior y duración, se escoge la tarea con menor orden lexicográfico inicial.

Destacamos que, si se modifica la ordenación utilizada, ya sea mediante la modificación de los criterios establecidos o la inclusión de nuevos criterios, la solución ofrecida por el procedimiento puede variar. Es por ello que el tercer criterio se ha alterado para incluir una dosis de aleatoriedad en el procedimiento que permita la construcción de soluciones diversas, si se dispone de más tiempo de cálculo. Dicha aleatoriedad se consigue añadiendo un valor aleatorio comprendido entre 0 y 5 veces la duración máxima de una tarea al valor utilizado en dicho paso.

- *Unificación de estaciones con idéntica composición*. Para eliminar la construcción de estaciones idénticas, se obliga a que las tareas sean escogidas en orden no decreciente respecto al orden de la lista obtenida en la reordenación.

Lógicamente, el procedimiento busca sólo asignaciones formadas por tareas que no forman parte de la solución parcial original.

4.2. Metaheurística Beam Search

Básicamente, la implementación realizada de la metaheurística *Beam Search* realiza una búsqueda usando dos listas semejantes a las que se emplean en [11].

La primera lista corresponde a un nivel del árbol de búsqueda ya desarrollado y la segunda corresponde al nivel del árbol de búsqueda en construcción. Para cada solución parcial de la primera lista se genera un conjunto de soluciones parciales bajo el control de *beamextensions* y se almacenan en la segunda lista. Cuando la primera lista está vacía, la segunda lista se somete a un procedimiento de reducción hasta dejar un máximo de *beamwidth* soluciones parciales en la lista. Esta lista reducida es la que se utiliza como primera lista en la siguiente iteración. El algoritmo se detiene cuando una solución parcial incluye todas las tareas que deben asignarse, esto es una solución completa, o cuando no existen soluciones parciales para desarrollar (ver esquema en figura 1). Tras ello, se vuelve a iniciar la búsqueda de nuevas soluciones modificando la ordenación de tareas tal como se ha visto en la sección 4.1.

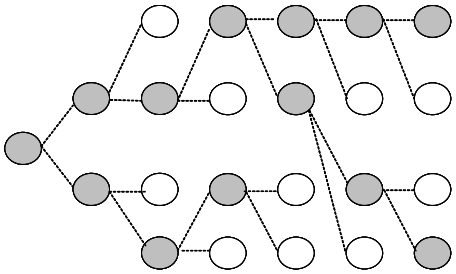

Figura 1. Esquema Beam Search. En cada nivel del árbol de búsqueda, se selecciona un conjunto de soluciones parciales (tareas asignadas a un número de estaciones que se corresponde con el nivel) cuyo cardinal es *beamwidth* como máximo; cada una de estas soluciones parciales se desarrolla (como máximo en *beamextensions* nuevas soluciones parciales) dando lugar a un nuevo conjunto en el nivel siguiente. Se concluye cuando se halla una solución que contiene todas las tareas.

De forma similar a la fase constructiva de estaciones, aquí se dispone de una técnica para determinar qué soluciones parciales se continúan investigando y se emplean diversas técnicas para mejorar el procedimiento. A continuación se detallan las técnicas utilizadas:

- Para determinar si las soluciones parciales pueden llevar a una mejora de la solución conocida, se construye una solución inicial mediante el algoritmo original de Hoffmann, que puede verse como un caso especial de nuestra heurística con *beamwidth* y *beamextensions* iguales a 1.

- A cada solución parcial construida se le aplica la cota LB1 considerando las tareas que aún no forman parte de la solución y se determina si dicha solución parcial puede conducir a una solución con menos estaciones; si no es así, se descarta dicha solución parcial.

- Se determina si dos soluciones parciales construidas en aplicaciones del procedimiento de Hoffmann son equivalentes. Dos soluciones parciales equivalentes son aquéllas que tienen el mismo conjunto de tareas asignadas. En caso de encontrar dos soluciones parciales equivalentes, se descarta una de las dos.

- Las soluciones candidatas se ordenan según la suma de tiempos libres de todas sus estaciones en orden no decreciente, seleccionando las *beamwidth* mejores para la siguiente etapa.

Cabe citar que el tercer punto indicado se aleja del enfoque *Beam Search*, acercándose a una exploración no basada en un árbol, sino en un grafo más semejante a la Programación Dinámica. El procedimiento implementado guarda mucha similitud con la Programación Dinámica Acotada, véase [2], donde se utilizan un conjunto de reglas parecido al establecido en el presente trabajo para explorar una parte del espacio de estados de un problema de secuenciación de unidades en contexto JIT.

El procedimiento antes descrito se aplica repetidamente hasta que se agota el tiempo concedido. La ordenación de tareas con un factor de aleatoriedad que se realiza en el procedimiento constructivo hace que las soluciones construidas no tengan por qué ser las mismas.

Finalmente, cabe citar que una iteración del algoritmo *Beam Search* no tiene por qué conducir a la construcción de una nueva solución, ya que el procedimiento se reinicia tan pronto como no quedan soluciones parciales que puedan mejorar la solución conocida.

5. Experiencia Computacional

Para comprobar la calidad del algoritmo propuesto se ha realizado una implementación en C++ utilizando el compilador GCC versión 4.0.1. El ordenador utilizado en esta experiencia computacional ha sido un Macintosh MacBookPro con un procesador 2.33 Ghz. Intel Core 2 Duo y 2 GB. de memoria RAM utilizando MAC OS X 10.4.9 como sistema operativo. Ni el compilador utilizado, ni el programa implementado, hacen uso de ningún procedimiento de paralelización del código, por lo que se hace uso únicamente de un procesador, y por tanto puede considerarse al ordenador utilizado, a efectos de esta experiencia computacional, como un ordenador con un único procesador a 2.33 Ghz. Para hacer uso de la propiedad de reversibilidad de las instancias que se ha expuesto en sección 3, el algoritmo resuelve cada instancia en sentido directo e inverso de forma separada, anotándose en la presente experiencia computacional el valor de la mejor solución obtenida por ambos procedimientos y la suma del tiempo utilizado por ambas ejecuciones por separado.

Las instancias utilizadas en la experiencia computacional corresponden a la colección disponible en la página web www.assembly-line-balancing-de, y que corresponden a las instancias utilizadas en la mayoría de trabajos de la literatura para comparar la calidad de las soluciones ofrecidas por los procedimientos propuestos. Esta colección cuenta con 269 instancias con un número de tareas entre 8 y 297, de las que se conoce la solución óptima para 268 de ellas.

El algoritmo presentado se compara con los dos mejores procedimientos disponibles en la literatura. El primero es el algoritmo SALOME [18], y el segundo el algoritmo Beam-ACO [6]. Los resultados de SALOME corresponden a los disponibles en la misma página web en que se encuentran las instancias, *www.assembly-line.balancing.de*, y en las que se ha utilizado un ordenador moderno con un procesador 3Ghz Intel Pentium IV y 1,5 GB. de memoria RAM, permitiendo un tiempo de ejecución igual a 3600 segundos. Los resultados del algoritmo Beam-ACO corresponden a la experiencia original realizada en el artículo citado en que no se especifica el ordenador utilizado pero sí que el algoritmo fue ejecutado en un total de 10 ocasiones con un tiempo concedido a cada una de las iteraciones igual a 120 segundos de tiempo de CPU, totalizando 20 minutos como máximo de tiempo de computación. Se supone que los dos ordenadores utilizados en las experiencias computacionales anteriores y el utilizado en la actual experiencia computacional tienen características semejantes y que la diferencia de ordenador o compilador utilizado son negligibles.

El algoritmo propuesto se ha probado con diferentes juegos de parámetros, *beamwidth*=(5, 10, 25) y *beamextensions*=(5, 10), permitiendo un tiempo de computación t=(1,5,10,30,60) segundos medido en tiempo real y no en tiempo CPU cuyo valor siempre es inferior al tiempo real. Cada juego de parámetros ha sido lanzado en una única ocasión.

Algoritmo	Tiempo (s)	#mejores
SALOME	3600	261
Beam-ACO	120 (x10)	245
Beam (5,5)	1	242
Beam (5,5)	5	251
Beam (5,5)	60	255
Beam (10,5)	5	256
Beam (10,5)	30	262
Beam (25,10)	10	261
Beam (25,10)	60	266

Tabla 1. Resultados de la experiencia computacional. Se detalla el número de mejores soluciones conocidas para los algoritmos SALOME y Beam-ACO, así como para el algoritmo presentado Beam Search con diferentes parámetros (*beamwidth*, *beamextensions*) y tiempos de ejecución concedidos.

La tabla 1 recoge los resultados obtenidos para algunas de las configuraciones probadas de *beamwidth*, *beamextensions* y *tiempo*. Se reporta el número de mejores soluciones obtenidas, que corresponden a soluciones óptimas en todos los casos menos en la instancia que queda abierta.

Como puede verse, los resultados del algoritmo son especialmente efectivos, incluso en tiempos de computación muy reducidos.

Con tiempos de computación de 5 segundo es suficiente para igualar los resultados de heurísticas como Beam-ACO, [6], y un incremento combinado de tiempo concedido y de los parámetros de control de la búsqueda, permite superar los resultados de la mejor heurística presente en la literatura y los obtenidos por SALOME con unos tiempos de computación muy inferiores. El algoritmo no obtiene el mejor resultado conocido para tres instancias, obteniendo una solución con una estación mas que la solución óptima conocida, que corresponden a la instancia con grafo de precedencias Barthold con tiempo de ciclo 85 y las instancias con grafo de precedencias Scholl y tiempos de ciclo 1515 y 1935.

6. Conclusiones

El presente trabajo presenta un procedimiento de resolución para el problema de equilibrado de líneas de montaje simple bajo el objetivo de minimizar el número de estaciones necesarias, SALBP-1. Los resultados obtenidos muestran que el algoritmo implementado es capaz de resolver 265 de las 269 instancias de la literatura de forma óptima, encontrando la mejor solución conocida para otra de las instancias. El procedimiento se basa en la metaheurística Beam Search, utilizando una serie de reglas adicionales para reducir búsqueda innecesaria, algunas de ellas inspiradas en la Programación Dinámica Acotada [2], y hace uso de un procedimiento constructivo interno que obtiene soluciones parciales de alta calidad en fracciones de segundo, aunque sus tiempos de computación crezcan (teóricamente) de manera exponencial en función del número de tareas.

Los resultados obtenidos mejoran todos los procedimientos anteriormente presentados para el problema. Cabe citar como ventaja adicional del procedimiento propuesto que es fácilmente adaptable a muchos de los casos generales estudiados en la literatura, teniendo únicamente que adaptar el procedimiento constructivo.

En la actualidad puede considerarse que el problema SALBP está resuelto, o al menos en cuanto se refiere a las instancias utilizadas como referencia en la literatura. Aún así los desarrollos para este problema pueden resultar de interés basándose en la aplicabilidad de estos métodos en la resolución de casos generales, los cuales, en general, son problemas abiertos.

7. Agradecimientos

Los autores agradecen la colaboración de Nissan Spanish Industrial Operations (NSIO) y a la Cátedra Nissan UPC por financiar parcialmente este trabajo. Esta investigación también ha sido financiada por el proyecto DPI2004-03475 del gobierno de España.

Referencias

[1] Agnetis, A., Ciancimino, A., Lucertini, M., Pizzichella, M. Balancing flexible lines for car components assembly, International Journal of Production Research 33, 333-350, 1995.

[2] Bautista, J., Companys, R., Corominas, A. Heuristics and exact algorithms for solving the Monden problem, European Journal of Operational Research 88, 101-113, 1996.

[3] Bautista, J., Pereira, J. Ant algorithms for a time and space constrained assembly line balancing problem, European Journal of Operational Research 177, 2016-2032, 2007.
[4] Baybars, I. A survey of exact algorithms for the simple assembly line balancing problem, Management Science 32, 909-932, 1986.
[5] Becker, C., Scholl, A. A survey on problems and methods in generalized assembly line balancing, European Journal of Operational Research 168, 694-715, 2006.
[6] Blum, C., Bautista, J., Pereira, J. Beam-ACO Applied to Assembly Line Balancing, Lectura Notes in Computer Science 4150, 96–107, 2006.
[7] Boysen, N., Fliedner, M., Scholl, A. A classification of assembly line balancing problems, European Journal of Operacional Research, disponible en: doi:10.1016/ j.ejor.2006.10.010.
[8] Buxey, G.M. Assembly line balancing with multiple Stations, Management Science 20, 1010-1021, 1974.
[9] Fleszar, K., Hindi, K.S. An enumerative heuristic and reduction methods for the assembly line balancing problem, European Journal of Operational Research 145, 606-620, 2003.
[10] Hoffmann, T.R. Assembly line balancing with a precedence matrix, Management Science 9, 551-562, 1963.
[11] Jackson, J.R. A computing procedure for a line balancing problem, Management Science 2, 261-271, 1956.
[12] Lapierre S.D., Ruiz, A., Soriano, P. Balancing assembly lines with tabu search, European Journal of Operational Research 168, 826-837, 2006.
[13] Martello, S., Toth, P. Knapsack problems – Algorithms and computer implementations, Wiley, New York, 1990.
[14] Miltenburg, J., Wijngaard, J. The U-line line balancing problem, Management Science 40, 1378-1388, 1994.
[15] Ow, P.S., Morton, T.E. Filtered Beam Search in Scheduling, International Journal of Production Research 26, 35-62, 1988.
[16] Salveson, M.E. The Assembly Line Balancing Problem. Journal of Industrial Engineering 6, 18-25, 1955.
[17] Scholl, A., Voß, S. Simple assembly line balancing - Heuristic approaches, Journal of Heuristics 2, 217- 244, 1996.
[18] Scholl, A., Klein, R. SALOME: A bidirectional branch and bound procedure for assembly line balancing, INFORMS Journal on Computing 9, 319-334, 1997.
[19] Scholl, A. Balancing and sequencing assembly lines, 2nd edition, Physica, Heidelberg, 1999.
[20] Scholl, A., Becker, C. State-of-the-art exact and heuristic solution procedures for simple assembly line balancing. European Journal of Operational Research 168, 666-693, 2006.
[21] Talbot, F.B., Patterson, J.H., Gehrlein, W.V. A comparative evaluation of heuristic line balancing techniques, Management Science 32,430-454, 1986.