Búsqueda de Vecindad Variable para el *Directed Circular Facility Layout Problem*

Daniel Duque[1], Eduardo G. Pardo[2], and Abraham Duarte[1]

[1] Departamento de Ciencias de la Computación, Universidad Rey Juan Carlos, Móstoles, España,
{d.duquem@alumnos.urjc.es, abraham.duarte@urjc.es}
[2] Departamento de Sistemas Informáticos, Universidad Politécnica de Madrid, Madrid, España,
{eduardo.pardo@upm.es}

Resumen En este trabajo se propone la aplicación de la metodología Búsqueda de Vecindad Variable, en concreto la variante *General Variable Neighborhood Search*, para la resolución de un nuevo problema de optimización, conocido como el *Directed Circular Facility Layout Problem*. El problema abordado consiste en la colocación de un conjunto de máquinas industriales de una cadena de trabajo, en disposición circular, de modo que se minimice la suma de los flujos entre las máquinas. En este problema, el flujo entre cada dos máquinas se considera dirigido en un solo sentido y dependerá del peso del mismo y de la distancia entre las máquinas. Para determinar la distancia entre las máquinas, no solo se tendrá en consideración su posición relativa, sino también el tamaño variable de las máquinas intermedias. El algoritmo propuesto ha sido evaluado sobre un conjunto de instancias del estado del arte y los resultados obtenidos comparados con los de un método exacto previo.

Keywords: Búsqueda de Vecindad Variable General, Ordenación Circular, Directed Circular Facility Layout Problem

1. Introducción

En este trabajo se aborda el problema conocido como *Directed Circular Facility Layout Problem* (DCFLP). El problema fue introducido en la literatura como tal por primera vez en [11] y se podría clasificar dentro de la categoría de problemas de optimización consistentes en encontrar una disposición circular de una serie de elementos, en este caso un conjunto de máquinas en un entorno industrial. Dada esta colocación circular, cada máquina genera un producto intermedio (o subproducto) de salida que será la entrada de otra máquina, no necesariamente contigua. En este sentido, dicho subproducto deberá ser desplazado de unas máquinas a otras en el sentido de las agujas del reloj. Es importante destacar que, este hecho, implica que si existiera, por ejemplo, un flujo entre una máquina y su máquina predecesora en la ordenación, debería recorrer toda la longitud completa de la circunferencia formada por el resto de máquinas. Para

abordar el DCFLP se propone en este trabajo la utilización de la metodología Búsqueda de Vecindad Variable, en concreto la variante *General Variable Neighborhood Search* (GVNS).

1.1. Estado del arte

Este problema pertenece al conjunto de problemas conocidos como *Facility Layout Problems* (FLP) que tratan sobre la colocación de un conjunto de máquinas en una instalación industrial, con una serie de restricciones y una determinada función objetivo. Según la clasificación de este tipo de problemas propuesta en [17], el DCFLP pertenecería al subconjunto de problemas con instancias especialmente estructuradas. Dentro de esta categoría se encuentran aquellas variantes que cuentan con diseños de manufactura especiales, tales como robots de transporte o cintas transportadoras, y que permiten el desplazamiento de los productos intermedios entre las máquinas. En el caso del DCFLP el desplazamiento es circular y los subproductos se mueven en un único sentido.

La aplicación real de este tipo de problemas (FLP) está enmarcada dentro de los conocidos como Sistemas de Producción Flexible, también conocidos por su acrónimo en inglés como FMS (*Flexible Manufacturing Systems*). La vinculación entre los FLP y los FMS ha sido estudiada en [14] y [9]. En estos trabajos se demuestra la influencia que puede llegar a tener una determinada distribución de un conjunto de máquinas en un proceso en cadena.

Existen otros problemas en la literatura que podrían ser clasificados dentro de la misma categoría que el DCFLP, donde se consideran también disposiciones circulares. En concreto se puede encontrar el *Directed Circular Arrangement Problem* (DCAP), problema que ha sido demostrado ser $\mathcal{NP}$-difícil [15] (lo que implica que el problema tratado en este trabajo también lo es), y el también $\mathcal{NP}$-difícil *Uni-directional Cyclic Layout Problem* (UCFLP) [13], incluyendo sus casos especiales: *Balanced Unidirectional Cyclic Layout Problem* (BUCFLP) y *Equidistant Unidirectional Cyclic Layout Problem* (EUCFLP).

Una de las principales diferencias entre las variantes anteriormente mencionadas y el DCFLP, abordado en este trabajo, es que este problema tiene en consideración la distancia entre cada par de máquinas, influyendo en dicha distancia el tamaño de las máquinas colocadas entre ambas. Otros problemas, como el UCFLP, anteriormente mencionado, trata únicamente con las distancias entre las ubicaciones de las máquinas, o el DCAP, que considera que todas las máquinas son de igual tamaño. En este sentido, el DCFLP tiene una gran similitud con la variante en donde las máquinas se disponen en una línea recta, en lugar de en disposición circular. Esta variante es conocida como el *Single Row Facility Layout Problem* (SRFLP) [18] y su vinculación con el DCFLP ha sido recogida en [11].

Por último, es importante destacar que en [11] se propone una reducción del problema al ampliamente conocido *Linear Ordering Problem* (LOP).

Para entender el planteamiento matemático de este problema, se parte de la versión más simple del problema (SRFLP), anteriormente descrita, donde la colocación de máquinas se realiza en una línea recta. En este caso, una instancia para

el SRFLP consiste en un conjunto de n máquinas, con longitudes $l_1, l_2, ..., l_n$ y conexiones c_{ij} entre pares de máquinas. El problema de optimización del SRFLP se formularía como sigue:

$$\min_{\pi \in \Pi_n} \sum_{i<j \in [n]} c_{ij} z_{ij}^{\pi} \tag{1}$$

donde Π_n son todas las permutaciones posibles de las máquinas, y z_{ij}^{π} la distancia entre los centros de las n máquinas i y j con respecto a una permutación $\pi \in \Pi_n$.

Para llegar a la formulación del DCFLP, basta con realizar una pequeña variación en la formulación presentada en la ecuación (1), quedando de la siguiente manera:

$$\min_{\pi \in \Pi_n} \sum_{i,j \in [n], i \neq j} f_{ij} z_{ij} \tag{2}$$

donde ahora f_{ij} es el flujo existente entre las máquinas i y j. También habría que tener en cuenta el nuevo cálculo de las longitudes entre máquinas, aplicando la peculiaridad del sentido único que ya se ha descrito. Por otra parte, se establece además que la máquina 1 siempre estará en la primera posición, con el objetivo de reducir la dimensión del problema y evitar la simetría.

1.2. Ejemplo de instancia y solución

Una vez presentado el problema y su motivación, se mostrará un ejemplo de instancia así como una posible solución del mismo para facilitar su comprensión. Este ejemplo fue introducido previamente en [11]. En la Figura 1 se presenta, representada esquemáticamente, una instancia para el DCFLP.

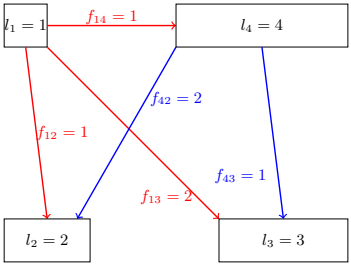

Figura 1. Ejemplo de instancia de 4 máquinas

La instancia está compuesta por cuatro máquinas etiquetadas con un identificador y su longitud asociada: $l_1 = 1$, $l_2 = 2$, $l_3 = 3$, y $l_4 = 4$. Además, en

la figura, se representan mediante flechas los siguientes flujos entre máquinas: $f_{12} = f_{14} = f_{43} = 1$, y $f_{13} = f_{42} = 2$.

Dado este ejemplo particular, una posible disposición óptima de las máquinas para el DCFLP sería 1–3–4–2, representada en la Figura 2. Como se puede observar, las máquinas están dispuestas a lo largo de una circunferencia, estando la máquina número 1 colocada en la primera posición, la máquina número 3 en la segunda posición y así sucesivamente. Se indica también en la figura el sentido en el que deben considerarse los flujos. En este caso el coste de la función objetivo de esta solución I es 30,5 obtenido de la siguiente manera:

$$fo(I) = 2 * 2 + 5{,}5 * 1 + 8{,}5 * 1 + 3 * 2 + 6{,}5 * 1 = 30{,}5$$

donde el primer elemento de cada producto representa la distancia entre cada par de máquinas y, el segundo, el flujo entre las mismas.

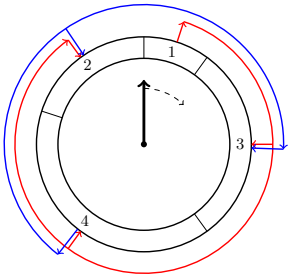

Figura 2. Solución óptima para la instancia de la Figura 1

En lo que resta de documento se presentan en profundidad los algoritmos propuestos para abordar el problema (Sección 2). Además, en la Sección 3 se introducen las instancias utilizadas para los experimentos, las pruebas llevadas a cabo para ajustar los parámetros de los algoritmos, y la comparación con el estado del arte. Finalmente, en la Sección 4 se exponen las conclusiones del trabajo.

2. Descripción algorítmica

En este trabajo se propone la utilización de la metodología Búsqueda de Vecindad Variable, también conocida como VNS, por su acrónimo en inglés del

término *Variable Neighborhood Search*. Puede encontrarse extensa documentación acerca de esta metodología en [8], donde aparece una amplia descripción de las ideas básicas de la misma así como de las extensiones más conocidas. En concreto, VNS ofrece una técnica de escape de óptimos locales basada en el cambio sistemático de estructuras de vecindad durante la búsqueda de soluciones. La metodología contempla fases de intensificación (mediante la utilización de búsquedas locales para un entorno determinado) y de diversificación (mediante la utilización de operaciones de agitación para escapar de dichos óptimos locales y explorar nuevas regiones del espacio de soluciones).

La metodología VNS, sin embargo, no establece cómo deben construirse las soluciones iniciales o qué punto de partida tomar. En ocasiones es habitual encontrar algoritmos basados en VNS que parten de una solución aleatoria. En este caso, como punto de partida para VNS, se han utilizado las soluciones obtenidas con un algoritmo *Greedy Randomized Adaptive Search Procedure* (GRASP). Esta metodología fue presentada por primera vez en [7] y está descrita en profundidad en [16]. GRASP se puede catalogar como una metaheurística constructiva multiarranque de dos fases. En la primera fase, GRASP construye una solución y, en la segunda fase, la mejora con un método de búsqueda local. En la fase de construcción, se forma la solución iterativamente guiándose por una función voraz (*greedy*), pero ligeramente aleatorizada (*randomized*). Los elementos candidatos a ser añadidos a la solución en la siguiente iteración son adaptativos, ya que se van modificando conforme se va construyendo la solución (*adaptive*).

El algoritmo GRASP propuesto en este trabajo sigue un esquema similar al presentado en el Algoritmo 1. En particular, el algoritmo se ejecuta durante un número máximo de iteraciones (delimitado por el parámetro de entrada `máx`). En cada iteración, comienza con la fase de construcción, asignando la primera máquina en la primera posición e introduciendo el resto de máquinas en el conjunto de candidatas. A continuación, a cada paso de la fase de construcción, evalúa todas esas máquinas candidatas con respecto a un determinado criterio voraz (mediante la función `Val`). Esto permite obtener una ordenación de las mismas, si fueran la siguiente máquina en ser añadida a la solución. Cada máquina evaluada se inserta en la lista de candidatas (CL), y se escoge un porcentaje de las mejores candidatas, determinado por el parámetro `alpha`, para construir la lista restringida de candidatas (RCL). Por último, se selecciona una máquina al azar dentro de la RCL y se le asigna la siguiente posición disponible (`i`). La máquina asignada, será entonces borrada de la lista de candidatas para la siguiente iteración del algoritmo. Una vez construida la solución, el algoritmo pasa a la segunda fase, la búsqueda local, con el objetivo de alcanzar un óptimo local desde la solución de partida. Tras la fase de búsqueda local, la solución S es comparada con la mejor solución encontrada hasta el momento y, en caso de ser la mejor, guardada para compararla con las mejores soluciones obtenidas en posteriores iteraciones. El algoritmo acaba devolviendo la mejor solución encontrada después del número preestablecido de iteraciones máximas.

Una vez construida una solución, se tratará de mejorarla por medio de una variante de VNS, en concreto la variante GVNS (del inglés *General Variable*

Algoritmo 1: Esquema general de GRASP

```
   Entrada: instancia I, valor alpha, valor máx
   Salida: solución mejor
 1 para iteración = 1 → máx hacer
 2     Añadir(S, máquina 1, 1);
 3     candidatas ← resto de máquinas;
 4     para i = 2 → longitud hacer
 5         para cada máquina ∈ candidatas hacer
 6             val ← Val(máquina);
 7             CL ← CL ∪ (máquina, val);
 8         fin
 9         RCL ← Seleccionar(alpha, CL);
10         elegida ← ObtenerAleatorio(RCL);
11         Añadir(S, elegida, i);
12         candidatas ← candidatas \ elegida;
13     fin
14     BúsquedaLocal(S);
15     mejor ← Comparar (mejor, S);
16 fin
17 devolver mejor
```

Neighborhood Search). Puede encontrarse un pseudocódigo de esta variante en Algoritmo 2. En concreto, mientras que no se alcance la condición de parada (tiempo, número máximo de iteraciones, etc.) el algoritmo comienza explorando la vecindad más próxima (`k=1`). Las variantes basadas en VNS utilizan el mecanismo de perturbación de la solución de partida para escapar de óptimos locales y poder así alcanzar otros. Por ello, lo primero que se realiza es la operación de `Shaking`, en la que la solución es perturbada en función del tamaño de `k` (que indica la vecindad explorada). En concreto, en la configuración de este algoritmo, la perturbación consistirá en movimientos de intercambio de máquinas al azar, donde cada movimiento consiste en seleccionar dos máquinas e intercambiar su posición actual. A continuación, la solución perturbada pasa a ser el parámetro de entrada de un método VND (del inglés *Variable Neighborhood Descent*). Este algoritmo, explora diferentes vecindades, de manera intensiva, y devuelve una solución que es óptimo local con respecto a todas las vecindades exploradas. En este caso se han utilizado dos búsquedas locales para explorar dichas vecindades: una búsqueda local basada en movimientos de inserción (una máquina es eliminada de su posición actual e insertada en otra) y otra búsqueda local basada en movimientos de intercambio (anteriormente descritos). Ambas búsquedas locales siguen una estrategia tipo *best improvement*. Por último, el método `NeighborhoodChange` es el encargado de guardar los cambios realizados, si estos han mejorado la solución de partida, o de revertirlos en caso contrario. En caso de mejora, el algoritmo comienza de nuevo desde la primera vecindad (`k`=1) y, en caso contrario, continúa con la siguiente vecindad (`k`=`k`+1).

Algoritmo 2: Esquema general de GVNS

Entrada: Una instancia I y una solución S
Salida: Una solución S' mejor o igual que S

```
1 mientras !(condición de parada) hacer
2 |  k ← 1;
3 |  mientras k ≤ kmax hacer
4 |  |  S' ← Shaking(S, k);
5 |  |  S'' ← VND(S');
6 |  |  k ← NeighborhoodChange(S,S'', k);
7 |  fin
8 fin
9 devolver S''
```

3. Resultados experimentales

En esta sección se realiza la evaluación de los algoritmos propuestos. En la Sección 3.1 se recogen los conjuntos de instancias utilizados. En la Sección 3.2 se presenta el ajuste de parámetros para GRASP y GVNS. Por último, en la Sección 3.3 se presenta una comparación con un algoritmo exacto previo.

3.1. Instancias

En concreto, se ha seleccionado un conjunto de 98 instancias, presentadas originalmente en [11] y que han sido previamente utilizadas en las variantes SRFLP y DCAP, así como otras utilizadas en el contexto del problema *Minimum Weighted Linear Arrangement Problem*.

Dada la singularidad de esta variante, donde existe un único flujo entre cada par de máquinas y que además este flujo debe ser dirigido, es decir, de una máquina hacia otra y no al revés, será necesario realizar una adaptación de las instancias anteriormente mencionadas. En particular, existen dos opciones a la hora de realizar la adaptación de las instancias, lo que hará que cada instancia pueda ser: *one-way*, caso en el que se considera que el flujo va siempre en un mismo sentido, es decir, $f_{ij} := c_{ij}$ y $f_{ji} := 0$, o *random*, cuando se considera que el flujo puede ir en cualquier sentido de una misma conexión y este se determina al azar entre $f_{ij} := c_{ij}$ y $f_{ji} := 0$ o $f_{ij} := 0$ y $f_{ji} := c_{ij}$.

Antes de pasar a comparar los resultados obtenidos por el agoritmo propuesto, con los del estado del arte, es necesario ajustar algunos parámetros del mismo. Para ello se han llevado a cabo una serie de experimentos preliminares tanto con GRASP como con GVNS (ver Sección 3.2). Estos experimentos han sido ejecutados sobre una selección de 10 instancias adaptadas del modo *one-way*.

3.2. Ajuste de parámetros de GRASP y GVNS

El algoritmo GRASP recibe dos parámetros de entrada, el número de iteraciones y el valor alpha, que ayuda a determinar el tamaño de la lista de candidatos restringida. Para el primer parámetro se tomará un valor típico en la metodología GRASP de 100 iteraciones. En cuanto al segundo parámetro se refiere, alpha, se han evaluado un conjunto de valores que representan un porcentaje del total de máquinas disponibles para ser añadidas a la solución. En concreto, dicho porcentaje se determina con respecto al tamaño de la instancia (número de máquinas) aunque, siempre que sea posible, se establecerá un valor mínimo de 5 máquinas en la lista restringida de candidatos. Es importante destacar que dado que el parámetro alpha afecta únicamente a la fase de construcción, las soluciones obtenidas en este experimento se han alcanzado sin efectuar la fase de búsqueda local de GRASP, tras 100 ejecuciones de la fase de construcción. En concreto, se han llevado a cabo pruebas con valores de 10 %, 20 %, 30 %, 40 % y 50 % del tamaño de las instancias, como porcentaje de elementos a ser añadidos a la RCL. Si bien la desviación más baja y la más alta difieren en menos de 1,2 puntos, las mejores soluciones se obtienen con un valor de 30 % para este parámetro. Por lo tanto, se elegirá este valor para futuros experimentos.

Una vez definido el parámetro clave del algoritmo GRASP, se procede a ejecutar la hibridación con GVNS descrita en la sección previa y que consiste en ejecutar GVNS tomando como punto de partida la mejor solución obtenida por GRASP. En concreto, el algoritmo GVNS deberá ajustar el parámetro k_{max}, que determina el máximo número de vecindades a explorar. El valor de k_{max} será ajustado como un porcentaje con respecto del tamaño de la instancia. Se ha ejecutado GVNS sobre la mejor solución obtenida después de ejecutar GRASP durante 50 y 100 iteraciones. La desviación respecto al óptimo con 50 iteraciones es muy pareja a la de 100 iteraciones, consumiendo aproximadamente la mitad de tiempo. Además, se han probado valores de k_{max} equivalentes 5 %, 10 %, 15 %, 20 % y 25 % del tamaño de la instancia. Se observa que cuando el valor k_{max} corresponde al 25 %, se alcanzan las mejores soluciones. Por lo tanto se tomará el valor del 25 % para k_{max}, así como 50 iteraciones como referencia, para el resto de experimentación.

3.3. Comparación con el estado del arte

Una vez realizada la experimentación previa, el algoritmo, con las configuraciones adecuadas debe compararse con métodos previos del estado del arte. En particular, el algoritmo propuesto es una hibridación del algoritmo GRASP mejorado con GVNS. La configuración de los parámetros del algoritmo consiste en una ejecución de 50 iteraciones de GRASP, en la que el tamaño de la RCL será el 30 % del tamaño de la instancia. Una vez ejecutada la fase constructiva de GRASP se tomará la mejor solución obtenida y, sobre ella, se aplicará GVNS, con un valor de k_{max} que será el 25 % del tamaño de la instancia evaluada.

En la Tabla 1 se recogen, los resultados obtenidos por el algoritmo propuesto para las adaptaciones *one-way* y *random*, de las instancias consideradas. En particular, las instancias han sido agrupadas en cuatro bloques y, por lo tanto, los

resultados presentados son promedios. La agrupación de las instancias se ha realizado de la siguiente manera: instancias SRFLP entre 8 y 36 máquinas, instancias SRFLP entre 40 y 56 máquinas, instancias SRFLP entre 60 y 80 máquinas, e instancias DCAP de múltiples tamaños. Para cada conjunto se reporta tanto la desviación con respecto a la mejor solución concida (ya sea aquella alcanzada por el agoritmo exacto propuesto en [11] o bien, en caso de que este no sea capaz de alcanzar el óptimo, aquella obtenida por una heurística). A modo de resumen, la desviación de las funciones objetivo obtenidas mediante el algoritmo propuesto, respecto a las obtenidas por el algoritmo exacto son muy parecidas para los conjuntos SRFLP (menos del 1 %) con una reducción en el tiempo de ejecución del más del 90 %, lo que hace pensar la utilidad de la heurística propuesta en este contexto. Sin embargo, los resultados son ligeramente peores para el conjunto de instancias DCAP, donde el algoritmo propuesto se ha quedado a más de un 5 % y 9 % respectivamente. De igual modo, se puede ver como las instancias con una adaptación *one-way* parecen ligeramente más sencillas de resolver que aquellas de tipo *random*.

	one-way		random	
Intancias	Desv.(%)	Ahorro tiempo (%)	Desv.(%)	Ahorro tiempo (%)
SRFLP 8-36	0,03565	90,34	0,00435	93,13
SRFLP 40-56	0,04055	95,31	0,50327	99,35
SRFLP 60-80	0,00446	97,89	0,48352	98,42
DCAP	5,20754	94,82	9,68348	95,28

Tabla 1. Resumen de resultados para instancias *one-way* y *random*

4. Conclusiones

En este trabajo se ha propuesto la hibridación de las metodologías GRASP y VNS. En concreto se ha utilizado GRASP como fase constructiva de las soluciones y VNS como fase de mejora posterior. Para la fase de mejora se ha propuesto un algoritmo basado en la variante GVNS, considerando dos búsquedas locales: una de intercambios y otra de inserciones. El algoritmo propuesto ha sido evaluado sobre un *dataset* compuesto por diversos conjuntos de instancias, utilizadas previamente en el estado del arte. Se ha comparado el algoritmo con el método exacto propuesto en [11]. Como resultado, el algoritmo desarrollado es capaz de obtener soluciones muy parecidas a las obtenidas por el método exacto, pero con un considerable ahorro de tiempo de más del 90 %. La aplicación de algoritmos heurísticos en este problema respondería a la necesidad de adaptar industrias flexibles periódicamente, ya que si el conjunto de máquinas va a estar fijo durante mucho tiempo, convendría utilizar algoritmos exactos para garantizar una solución óptima.

Agradecimientos

Este trabajo ha sido financiado por el Ministerio de Economía y Competitividad (TIN2015-65460-C2-2-P) y por la Comunidad de Madrid (S2013/ICE-2894).

Referencias

1. Amaral A. R. S., *On the exact solution of a facility layout problem*, European Journal of Operational Research, 173(2):508-518, 2006.
2. Amaral A. R. S., *An exact approach to the one-dimensional facility layout problem*, Operations Re-search, 56(4):1026-1033, 2008.
3. Amaral, A.R.S., *A new lower bound for the single row facility layout problem*, Discrete Applied Mathematics, 157(1):183-190, 2009.
4. Anjos M. F., A. Kennings and A. Vannelli, *A semidefinite optimization approach for the single-row layout problem with unequal dimensions*, Discrete Optimization, 2(2):113-122, 2005.
5. Anjos M. F., and A. Vannelli, *Computing Globally Optimal Solutions for Single-Row Lay-out Problems Using Semidefinite Programming and Cutting Planes*, INFORMS Journal On Computing, 20(4):611-617, 2008.
6. Anjos M. F. and G. Yen, *Provably near-optimal solutions for very large single-row facility layout problems*, Optimization Methods and Software, 24(4):805-817, 2009.
7. Feo T. A. and M. G. C. Resende, *Greedy Randomized Adaptive Search Procedures*, Journal of Global Optimization, 6(2):109-133, 1995.
8. Hansen P., N. Mladenović, J. Brimberg and J. A. Moreno Pérez, *Variable neighborhood search: Principles and applications*, Hanbook of Metaheuristics (2nd Edition), 61-86, 2010.
9. Hassan M. M. D., *Machine layout problem in modern manufacturing facilities*, International Journal of Production Research, 32(11):2559-2584, 1994.
10. Heragu S. S. and A. Kusiak, *Efficient models for the facility layout problem*, European Journal of Operational Research, 53(1):1-13, 1991.
11. Hungerländer P., *A New Modelling Approach for Cyclic Layouts and its Practical Advantages*, Alpen-Adria Universität Klagenfurt, 2013
12. Hungerlander P. and F. Rendl, *A computational study and survey of methods for the single-row facility layout problem*, Computational Optimization and Applications, 55(1):1-20, 2013.
13. Kouvelis P. and M. W. Kim, *Unidirectional loop network layout problem in automated manufacturing systems*, Operations Research, 40(3):533-550, 1992.
14. Leung J., *Polyhedral structure and properties of a model for layout design*, European Journal of Operational Research, 77(2):195-207, 1994.
15. Liberatore V., *Circular arrangements*, Automata, Languages and Programming, 1054-1065, Springer 2002.
16. Martí R., J. M. Moreno-Vega and A. Duarte, *Advanced Multi-start Methods*, Hanbook of Metaheuristics (2nd Edition), 265-281, 2010.
17. Meller R. and K.-Y. Gau. *The facility layout problem: Recent and emerging trends and perspectives*, Journal of Manufacturing Systems, 5(5):351-366, 1996.
18. Rubio-Sánchez, M., M. Gallego, F. Gortázar and A. Duarte *GRASP with path relinking for the single row facility layout problem*, Knowledge-Based Systems, In press, 2015.
19. Simmons D. M., *One-Dimensional Space Allocation: An Ordering Algorithm*, Operations Research, 17:812-826, 1969.