# GRASP con Reencadenamiento de Trayectorias para el Single Row Facility Layout Problem

Manuel Rubio-Sánchez, Micael Gallego, Francisco Gortázar, Abraham Duarte

Escuela Técnica Superior de Ingeniería Informática de la Universidad Rey Juan Carlos
{manuel.rubio, micael.gallego, francisco.gortazar, abraham.duarte}@urjc.es

**Abstract.** El single row facility layout (SRLFP) es un problema NP duro que consiste en encontrar la disposición óptima de un conjunto de instalaciones de igual altura pero diferente anchura, disponiéndolas unas a continuación de las otras en línea. En este trabajo se presenta un algoritmo basado en GRASP con Reencadenamiento de Trayectorias para el SRFLP. Se presenta también un nuevo conjunto de instancias, más grandes y difíciles. Los resultados obtenidos y los tests estadísticos efectuados permiten concluir que el método desarrollado mejora en calidad y tiempo a los métodos del estado del arte.

**Keywords.** Single row layout facility problem · Greedy Randomized Adaptive Search Procedure · Path Relinking

## 1 Introducción

El SRFLP es un problema NP duro que consiste en encontrar la disposición óptima de un conjunto de instalaciones rectangulares cuando se disponen en línea unas junto a otras. Este problema tiene aplicaciones prácticas como la disposición de habitaciones a lo largo de un pasillo, la ubicación de libros en estantes o el almacenamiento de información en discos magnéticos. El objetivo es encontrar una ordenación de dichas instalaciones que minimice la suma ponderada de las distancias entre los centros de todos los pares de instalaciones. $F = \{1,2,\dots,n\}$ es un conjunto de $n > 2$ instalaciones rectangulares de alto fijo y diferentes longitudes $l_i > 0, \forall i \in F$. El peso entre las instalaciones $i$ y $j$ es $c_{ij} = c_{ji} > 0, \forall i,j \in F$. Una solución al problema es una ordenación $\pi =< \pi(1), \pi(s), \dots, \pi(n) >$ de las instalaciones en $F$, cuyo coste $C(\pi)$ se define como sigue:

$$C(\pi) = \sum_{1 \le q < r \le n} c_{\pi(q)\pi(r)} \cdot d_{\pi(q)\pi(r)}, \quad (1)$$

Donde $d_{\pi(q)\pi(r)}$ representa la distancia entre los centros de las instalaciones $\pi(q)$ y $\pi(r)$ (dispuestas en la ordenación $\pi$ en las posiciones $q$ y $r$ respectivamente), y calculada como:

$$d_{\pi(q)\pi(r)} = \frac{l_{\pi(q)}}{2} + \sum_{q<s<r} l_{\pi(s)} + l_{\pi(r)} \,/2 \quad (2)$$

El problema de optimización consiste entonces en encontrar una ordenación $\pi^*$ que minimice (1).

En este artículo proponemos una adaptación de un método GRASP [1] para buscar de forma eficiente soluciones de alta calidad para el SRFLP. Se introduce un método constructivo que permite balancear entre intensificación y diversificación y un método de mejora muy rápido basado en hibridizar las búsquedas del mejor y el primero encontrado. Como procesado posterior introducimos el reencadenamiento de trayectorias con el objeto de encontrar nuevas soluciones de calidad.

## 2 Estado del arte

Dentro de los problemas de localización de instalaciones, el SRFLP es uno de los más activos actualmente. Existen estudios sobre el estado del arte del problema como los de Kothari y Ghosh [2] y Keller y Bucher [3]. El problema ha sido abordado utilizando algoritmos exactos (se conocen los óptimos de aquellas instancias con hasta 42 instalaciones) y heurísticos. Dentro de éstos últimos los trabajos más recientes y que ofrecen los mejores resultados son los de Kothari y Ghosh [4,5]. El primer trabajo presenta un algoritmo memético donde tanto las mutaciones como la búsqueda local se basan en movimientos de inserción. En el segundo trabajo los autores aplican la metodología de búsqueda dispersa al problema, incluyendo en dicho método algunos componentes del primer trabajo. Concretamente, el método de intensificación es el mismo y la combinación utiliza el mismo operador de *crossover* que se utiliza en el algoritmo genético. Como novedades, se introducen dos métodos de diversificación que se utilizan para generar el conjunto de referencia y un método adicional de combinación. Ambos métodos son capaces de obtener 41 y 42 de los mejores valores conocidos de un total de 43 instancias, lo que es un indicativo de que son necesarias instancias más difíciles para poder medir adecuadamente la calidad de los algoritmos desarrollados para este problema.

## 3 GRASP

La metodología GRASP fue desarrollada en los años 80 por Feo y Resende [1]. Se trata de una metodología con rearranque donde cada iteración consiste en dos pasos: una construcción voraz, ligeramente aleatorizada y adaptativa de una solución; una mejora de la solución construida hasta alcanzar un óptimo local. Estos dos pasos se repiten hasta que se alcanza la condición de parada, que puede ser un tiempo máximo de computación o un número máximo de soluciones construidas.

### 3.1 Método de construcción

Un procedimiento GRASP construye una solución $\pi$ de tamaño $n$ paso a paso insertando una instalación cada vez en una solución parcial $\pi_p$. La elección de cada instalación a insertar en cada paso utiliza decisiones voraces y aleatorizadas. De cara

a seleccionar la siguiente instalación a insertar en la solución parcial, medimos el coste de dicha solución parcial con la nueva instalación insertada. Formalmente:

$$C(\pi_p) = \sum_{1 \le q < r \le m} c_{\pi_p(q)\pi_p(r)} \cdot d_{\pi_p(q)\pi_p(r)}. \quad (3)$$

En base a (3), la mejor localización para una nueva instalación es aquella que minimiza $C(\pi_p)$. Para nuestro algoritmo constructivo, utilizamos una estrategia diferente a la habitual a la hora de seleccionar el siguiente candidato a insertar en la solución, propuesta por Resende y Werneck [6]. Primero ordenamos las instalaciones que aún no han sido insertadas en la solución parcial $\pi_p$ de acuerdo a estos costes. Estas instalaciones conforman lo que en GRASP se denomina la lista de candidatos (*CL*). Entonces se escogen $size$ elementos aleatoriamente de la *CL*, y de entre esta lista de candidatos restringida (*RCL*) se escoge el mejor en términos de su función de coste y se inserta directamente en la solución en la mejor posición posible.

### 3.2 Búsqueda local

La segunda fase de un algoritmo GRASP consiste en mejorar la solución construida con un método de búsqueda local. La idea consiste en aplicar alguna estrategia que modifique progresivamente el orden de las instalaciones de forma que el coste de la función objetivo se reduzca. Este proceso de búsqueda, conocido como operador de movimiento, es repetido hasta que no es posible mejorar más (es decir, guía la búsqueda hacia un óptimo local).

En particular, para el SRFLP se ha definido un tipo de movimiento conocido como *insert*. Concretamente, dada una solución $\pi$, el movimiento $insert(\pi, q, r)$ consiste en eliminar una instalación $\pi(q)$ de su posición actual $q$ e insertarla en la posición $r$. La nueva solución $\pi'$ depende de los valores relativos de $q$ y $r$. En este caso, el vecindario asociado tiene un tamaño de $n(n-1)$.

Existen dos estrategias típicas para explorar un vecindario: la mejor mejora (*best improvement*) y la primera mejora (*first improvement*). La primera explora todas las soluciones de la vecindad mediante un procedimiento determinista, y el mejor movimiento se aplica en cada iteración. La segunda (primera mejora) explora la vecindad de una solución inicial y realiza el primer movimiento que mejora la función objetivo. El procedimiento normalmente elije diferentes movimientos al azar para obtener soluciones diversas y finaliza cuando el vecindario completo ha sido explorado y ningún movimiento es capaz de mejorar el coste de la solución inicial.

Nosotros presentamos un nuevo método de búsqueda local híbrido, denominado LS-HYBRID, que usa movimientos de inserción y combina las estrategias de primera mejora (*first*) y mejora mejora (*best*). Dada una solución particular, la idea consiste en seleccionar una instalación aleatoriamente en cada paso (de forma similar a la estrategia de primera mejora), y posteriormente encontrar el mejor movimiento sólo para esa instalación (tal y como indica la estrategia de mejor mejora). Si el valor de función objetivo no mejora, el algoritmo selecciona aleatoriamente una instalación diferente y busca el mejor movimiento de inserción para ella. Este proceso es repetido mientras encuentre mejores soluciones. Finalmente, el algoritmo alcanza un óptimo

local y se detiene cuando no se puede realizar ningún movimiento de mejora que pueda mejorar el coste de la solución previa.

Con el fin de realidad la búsqueda local más eficientemente, el método analiza intercambios de posiciones consecutivas para encontrar el mejor movimiento, en vez de insertar una instalación directamente en la posición objetivo. Concretamente, dada una instalación $\pi(q)$, en la posición $q$, nosotros la intercambiamos con la instalación en la posición $q-1$, guardando el cambio asociado en la función objetivo (denominado como *valor del movimiento*). Posteriormente, intercambiamos las instalaciones en las posiciones $q-1$ y $q-2$, guardando de nuevo el valor asociado al movimiento. El método procede de forma similar hasta que alcanza la posición 1. Simétricamente, la búsqueda local realiza movimientos de intercambio entre posiciones consecutivas $q+1$ hasta $n$. La búsqueda local híbrida que proponemos calcula los valores de los movimientos de forma incremental ya que la evaluación de un movimiento de intercambio entre posiciones consecutivas se puede calcular en tiempo lineal. Concretamente, dada una solución $\pi$ y una instalación $\pi(q)$, el coste de una solución $\pi'$ después de realizar el movimiento $swap(\pi, q, q+1)$ se puede obtener como:

$$C(\pi') = C(\pi) + l_{\pi(q+1)}\left(\sum_{s=1}^{q-1} C_{\pi(s)\pi(q)} - \sum_{s=q+2}^{n} C_{\pi(q)\pi(s)}\right) + l_{\pi(q)}\left(\sum_{s=q+2}^{n} C_{\pi(q+1)\pi(s)} - \sum_{s=1}^{q-1} C_{\pi(s)\pi(q+1)}\right) \quad (4)$$

El primer término de (4) identifica el coste de la solución $\pi$. El segundo término ajusta el coste asociado con la instalación $\pi(q+1)$, mientras que el último término corrije el coste para $\pi(q)$. La combinación de la composición de movimientos de intercambio para obtener un movimiento de inserción general, junto con el cálculo incremental del coste tal y como se ha definido, hace que la búsqueda local sea muy eficiente.

## 4 Reencadenamiento de trayectorias

El reencadenamiento de trayectorias o *Path Relinking* (PR) es una metaheurística propuesta originalmente como una metodología para integrar estrategias de intensificación y diversificación en el contexto de la búsqueda tabú [7,8]. El método genera nuevas soluciones modificando iterativamente una solución inicial, transformándola en otras soluciónes de su vecindario, hasta que se alcanza una solución final usada como guía. Por tanto, el proceso construye una *trayectoria* de nuevas soluciones intermedias con la esperanza de que se pueda encontrar alguna mejor que las soluciones de alta calidad que están siendo conectadas.

La elección del operador de movimiento determina cómo se construye la ruta. En este artículo, para implementar la estrategia eficientemente, hemos usado una representación alternativa de la solución. Concretamente, si $\pi$ es una solución al

SRFLP (donde la instalación $\pi(q)$ está localizada en la posición $q$), entonces se puede especificar mediante su representación inversa $\pi^{-1}$, que indica la posición en la que se encuentran las instalaciones en $\pi$ (es decir, la instalación $i$ está localizada en la posición $\pi^{-1}(i)$ en $\pi$). Por ejemplo, sea $\pi_x = [2,3,1,4,5]$ una solución inicial y $\pi_y = [5,2,1,4,3]$ una solución guía. La representación inversa de $\pi_x$ es $\pi_x^{-1} = [3,2,5,4,1]$. Esta representación alternativa permite al algoritmo PR encontrar eficientemente una posición de una instalación.

Proponemos una estrategia, PR, más sofisticada relacionada con el cálculo de la distancia de Ulam [9], que mide el mínimo número de movimientos de inserción necesarios para transformar una ordenación en otra. Puede entenderse como *n* menos la longitud de la subsecuencia común más larga entre dos ordenaciones. El primer paso del PR consiste en identificar el conjunto de elementos más grande en ambas permutaciones que mantienen el mismo orden relativo entre ellas. Por ejemplo, sea $\pi_x = [3,7,8,4,1,2,5,6]$ y $\pi_y = [3,2,1,8,4,5,6,7]$ dos soluciones para alguna instancia del SRFLP. Sólo la instalación 3 está localizada en la misma posición (la primera) en ambas ordenaciones. No obstante, podemos observar que las instalaciones 3,8,4,5, y 6 comparten la misma ordenación relativa en ambas soluciones. En otras palabras, $\pi_x^{-1}(3) < \pi_x^{-1}(8) < \pi_x^{-1}(4) < \pi_x^{-1}(5) < \pi_x^{-1}(6)$ y $\pi_y^{-1}(3) < \pi_y^{-1}(8) < \pi_y^{-1}(4) < \pi_y^{-1}(5) < \pi_y^{-1}(6)$. Sea $\lambda$ este conjunto más largo de instalaciones, que no es único en general. Por conveniencia en la notación, lo representaremos como una lista ($\lambda = (3,8,4,5,6)$ en el ejemplo).

Nótese que si $\pi_x$ fuera la ordenación (1,2,..., *n*) entonces $\lambda$ podría corresponder a la mayor subsecuencia creciente de $\pi_y$. En cualquier caso, como $\pi_x$ puede ser cualquier ordenación, es necesario modificar las etiquetas de las instalaciones para obtener $\lambda$ calculando la subsecuencia creciente más larga. Concretamente, la instalación $\pi_x(1)$ podría renombrarse como 1, $\pi_x(2)$ como 2, y así para el resto. Esto se obtiene aplicando $\pi_x^{-1}$ a la ordenación inicial. Por tanto, $\lambda$ se puede obtener calculando primero la subsecuencia creciente más grande en la siguiente ordenación:

$$\tau = \pi_x^{-1}(\pi_y) = \left[\pi_x^{-1}\left(\pi_y(1)\right), \pi_x^{-1}\left(\pi_y(2)\right), \dots, \pi_x^{-1}\left(\pi_y(n)\right)\right]. \quad (5)$$

Nótese que $\tau$ codifica la localización en $\pi_x$ de las instalaciones en $\pi_y$ (es decir, $\tau(q) = \pi_x^{-1}\left(\pi_y(q)\right)$ es la posición de la instalación $q$-ésima en $\pi_y$). En otras palabras, indica el orden relativo en $\pi_x$ de las instalaciones en $\pi_y$. En el ejemplo $\tau = [1,6,5,3,4,7,8,2]$, ya que la primera instalación en $\pi_y$ está localizada en la primera posición en $\pi_x$, la segunda en $\pi_y$ está localizada en la sexta posición de $\pi_y$, y así sucesivamente.

Seguidamente, sea $\sigma$ la subsecuencia creciente más grande en $\tau$, con el valor (1,3,4,7,8) en el ejemplo (téngase en cuenta que los elementos en $\sigma$ no son necesariamente contiguos). La distancia de Ulam es la diferencia entre $n$ y el tamaño de $\sigma$ (que tiene el mismo tamaño que $\lambda$). En el ejemplo previo esto es: $8 - 5 = 3$, que significa que podemos transformar $\pi_x$ en $\pi_y$ realizando sólo 3 movimientos de inserción. Por tanto, PR podría generar una trayectoria con sólo 2 soluciones intermedias.

Finalmente, el algoritmo selecciona los movimientos de inserción que deben realizarse de forma aleatoria. En particular, sea $D$ el conjunto de instalaciones que no están en $\lambda$ (en el ejemplo, $D = \{1,2,7\}$). Primero, PR selecciona una instalación de $D$ aleatoriamente y la inserta en la posición en $\pi_x$ para incrementar el número de instalaciones que comparten el mismo orden relativo en ambas permutaciones. Por ejemplo, supongamos que PR selecciona la instalación 1. Ya que se debe insertar entre las instalaciones 3 y 8, puede ser insertada en la posición 2 o 3. Nuestro algoritmo propuesto selecciona uno de los posibles movimientos de forma aleatoria. El proceso se repite hasta que todos los elementos en $D$ han sido insertados.

## 5 GRASP con reencadenamiento de trayectorias

El reencadenamiento de trayectorias puede adaptarse en el contexto de GRASP como una estrategia de optimización posterior [10]. La combinación de ambos enfoques trabaja en un conjunto de soluciones de tamaño $b$ que normalmente se denomina conjunto élite (*elite set,* $ES = \{ES_1, ES_2, \dots, ES_b\}$). Habitualmente se ordena en base a la calidad de las soluciones (de la mejor, $ES_1$ a la peor $ES_b$). Primero se inicializa $ES$ con soluciones generadas por el procedimiento GRASP. A continuación, el algoritmo continua generando más soluciones que se combinan con aquellas en el conjunto élite usando el reencadenamiento de trayectorias. Las soluciones resultantes reemplazan a las soluciones existentes en el conjunto élite, mejorando la calidad total.

En nuestro procedimiento GRASP con reencadenamiento de trayectorias el conjunto élite se crea manteniendo las $b$ soluciones generadas por el procedimento GRASP (ver Sección 3) que es ejecutado $m \geq b$ veces. Para tener soluciones de calidad, $m$ es un valor grande que depende del tamaño del problema. A continuación el algoritmo itera generando una nueva solución GRASP $\pi_{\text{GRASP}}$, que será combinada con las soluciones del conjunto élite usando el procedimiento de reencadenamiento de trayectorias, hasta que se alcance un tiempo máximo de ejecución $T_{max}$. En concreto, en vez de generar una ruta entre $\pi_{\text{GRASP}}$ y una solución seleccionada del $ES$ (como es habitual), calculamos todas las trayectorias entre $\pi_{\text{GRASP}}$ y cada una de las soluciones del $ES$. Sea $\pi_{\text{PR}}$ la mejor solución encontrada en cada trayectoria, que ha sido optimizada adicionalmente usando alguno de los procedimientos de búsqueda local descritos en la sección 3.2. El algoritmo evalúa $\pi_{\text{PR}}$ para obtener cuándo debería incluirse en el conjunto élite. Primero, no puede estar ya incluida en el $ES$, y su coste asociado debe ser al menos tan grande como la peor solución en ES. Si se cumplen esas condiciones el algoritmo continua calculando la solución más cercana a $\pi_{\text{PR}}$ en el $ES$ (denominada $ES_j$). Formalmente, definimos la distancia como:

$$d(\pi_x, \pi_y) = \min\{\delta(\pi_x, \pi_y), \delta(\pi_x, \bar{\pi}_y)\} \tag{6}$$

donde $\bar{\pi}_y$ es la ordenación inversa de $\pi_y$ y:

$$\delta(\pi_x, \pi_y) = \sum_{i=1}^{n} \left|\pi_x^{-1}(i) - \pi_y^{-1}(i)\right| \tag{7}$$

es la „distancia de desviación“ [11]. Finalmente, $\pi_{\mathrm{PR}}$ se introduce en el conjunto élite, $ES_j$ se elimina, y $ES$ se reordena. No obstante, esto ocurre únicamente si $\pi_{\mathrm{PR}}$ es la mejor solución encontrada, o siempre que la solución para ser borrada no sea la mejor del conjunto élite (paso 13). De esa forma, si $\pi_{\mathrm{PR}}$ es mejor que $ES_1$ siempre se admite en el $ES$. Además, si es peor que el $ES_1$, entonces $ES_1$ no será borrada del conjunto élite.

## 6 Resultados computacionales

En esta sección se muestran los resultados de los experimentos computacionales realizados para comprobar la calidad de las estrategias propuestas. Todos los experimentos han sido realizados en un ordenador personal con un procesador Intel Core™ i7-4712HQ 3.3 GHz y 16 GB de RAM. Todo el código ha sido implementado en C y compilado con gcc.

Hemos ejecutado experimentos en 3 conjuntos de instancias usadas en la literatura del SRFLP. El primero, denominado *Anjos*, fue introducido en Anjos et al. (2005) y consiste en cuatro grupos de 5 instancias con $n$ =60,70,75 y 80. El segundo, denominado *sko*, fue introducido en [12]. Nosotros usamos 4 grupos de 5 instancias con $n$ =64,72,81 y 100. Finalmente, el tercer conjunto, denominado *Amaral*, fue presentado en [13] y consiste en 3 instancias con $n$ =110. Adicionalmente, hemos generado nuevas instancias más grandes para comparar los algoritmos. En concreto, hemos creado dos nuevos conjuntos, *Anjos-large* (40 instancias) y *sko-large* (40 instancias), donde los tamaños y los costes han sido obtenidas de forma que sean lo más parecidas a los conjuntos originales, pero más grandes. Cada conjunto contiene 5 instancias para cada uno de los valores $n$ =150, 200, 250, 300, 350, 400, 450 y 500. Estas instancias están disponibles en [14].

Los métodos del estado del arte usados para evaluar la calidad de nuestra propuesta han sido (i) el algoritmo genético GENALGO descrito en [4], y (ii) la variante de búsqueda dispersa (*scatter search*) SS-2P descrita en [5]. Para que la comparación sea lo más justa posible, hemos implementando esos métodos tal y como se describen en sus respectivos trabajos de presentación y usando las mismas técnicas que en el algoritmo propuesto.

La comparación con los métodos del estado del arte está dividida en dos experimentos diferentes. En el primer experimento, hemos comparado todos los métodos para el SRFLP (nuestra propuesta GRASP_PR, y los dos mejores métodos del estado del arte GENALGO y SS-2P) con los conjuntos de instancias *Anjos*, *sko* y *Amaral*. Los resultados de este experimento muestran que el algoritmo GRASP-PR, cuando se ejecuta una única vez durante $n/2$ segundos es capaz de obtener los mejores resultados para todas las instancias. En promedio, el tiempo de ejecución por instancia es de 38,83 segundos. Por otro lado, el algoritmo GENALGO es capaz de obtener los mismos resultados que GRASP_PR pero en mucho mayor tiempo de ejecución (2.723,86 segundos). Por último, el algoritmo SS-2P obtiene los mismos resultados que los otros dos métodos excepto en 3 instancias (sko_72_3, sko_81_3 y

sko_100_5) en un tiempo de ejecución en media ligeramente superior a GRASP_PR de 47,19. La conclusión que podemos obtener de este experimento es que estas instancias tan pequeñas del estado del arte no suponen un reto para los algoritmos

| Instance | GENALGO | SS-2P | GRASP-PR |
|---|---|---|---|
| Anjos-200-1 | 305497727 | **305461818** | **305461818** |
| Anjos-200-2 | **178807197.5** | 178852677.5 | 178816261.5 |
| Anjos-200-3 | 61892040 | 61891652 | **61891275** |
| Anjos-200-4 | 127743257 | 127736464 | **127735691** |
| Anjos-200-5 | 89059083.5 | 89141158.5 | **89057121.5** |
| Anjos-300-1 | 1550443050 | 1550822258 | **1549663657** |
| Anjos-300-2 | 955749527.5 | 957249994.5 | **955572066.5** |
| Anjos-300-3 | 308372773.5 | 308701657.5 | **308257630.5** |
| Anjos-300-4 | 603268519.5 | 603678289.5 | **602873168.5** |
| Anjos-300-5 | 466717889 | 466162354 | **466160264** |
| Anjos-400-1 | 5004496747.5 | 5005963158.5 | **5000752142.5** |
| Anjos-400-2 | 2916949529 | 2916726055 | **2910276759** |
| Anjos-400-3 | 921346203 | 922110067 | **921216455** |
| Anjos-400-4 | 1809520966 | 1809564367 | **1806061379** |
| Anjos-400-5 | 1404580939.5 | 1405152456.5 | **1402779472.5** |
| Anjos-500-1 | 12311278683 | 12304668851 | **12300409839** |
| Anjos-500-2 | 7505406907.5 | 7501448013.5 | **7493120635.5** |
| Anjos-500-3 | 2483990108 | 2483197897 | **2479333773** |
| Anjos-500-4 | 4288172936.5 | 4293459100.5 | **4285937468.5** |
| Anjos-500-5 | 3685342697.5 | 3680148707.5 | **3678038066.5** |
| sko-200-1 | 3233188 | 3231654 | **3231379** |
| sko-200-2 | 7758950 | 7758939 | **7758927** |
| sko-200-3 | 12740236 | 12741031 | **12739043** |
| sko-200-4 | 20263483 | 20268399 | **20260531** |
| sko-200-5 | 26873277.5 | 26873153.5 | **26871976.5** |
| sko-300-1 | 11278708 | 11263368 | **11251960** |
| sko-300-2 | 29008472 | 29061348 | **28993831** |
| sko-300-3 | 48828528.5 | 48894938.5 | **48800510.5** |
| sko-300-4 | 71303673.5 | 71259866.5 | **71194203.5** |
| sko-300-5 | 86834403.5 | 86806304.5 | **86792128.5** |
| sko-400-1 | 26780355 | 26754928 | **26718485** |
| sko-400-2 | 68058002 | 68046421 | **67996107** |
| sko-400-3 | 116150616 | 116088506 | **115942726** |
| sko-400-4 | 163359654 | 163371846 | **163099026** |
| sko-400-5 | 228119749.5 | 228115330.5 | **227778641.5** |
| sko-500-1 | 53030465 | 53117838 | **52951975** |
| sko-500-2 | 127631162 | 127648186 | **127428477** |
| sko-500-3 | 231465645.5 | 231408793.5 | **231041993.5** |
| sko-500-4 | 341175620 | 340848111 | **340390652** |
| sko-500-5 | 446498553 | 446641044 | **445738609** |

Tabla 1. Comparativa entre los métodos del estado del arte y el algoritmo GRASP-PR

(porque obtienen el mismo valor para la mayoría de ellas). Además, la diferencia en tiempo de ejecución de los algoritmos dificulta la comparación entre ellos. Por estos motivos, hemos diseñado un segundo experimento.

En el segundo experimento, usamos las 40 instancias de test con tamaños 200, 300, 400 y 500 en los conjuntos *Anjos-large* y *sko-large*. Para evaluar el rendimiento de los algoritmos hemos establecido el mismo tiempo de ejecución para los algoritmos en 1 hora por instancia. En la Tabla 1 se pueden ver los resultados obtenidos donde el mejor valor se resalta en negrita.

En este segundo experimento nuestro algoritmo GRASP_PR fue capaz de obtener mejores soluciones que los otros dos métodos en 39 de las 40 instancias. El algoritmo GENALGO obtuvo mejor solución que los demás métodos en una única instancia. Por último, el algoritmo SS-2R no fue capaz de superar nunca a los otros dos métodos, si bien fue capaz de obtener para una instancia una solución con el mismo valor en que la obtenida por el método GRASP_PR. Como conclusión, a la vista de los resultados obtenidos en este experimento, podemos considerar que el método GRASP_PR es mejor que los demás métodos.

Para soportar la afirmación previa, hemos aplicado un test de Friedman a los datos de la Tabla 1. Este test estadístico no paramétrico obtiene el *ranking* de cada método para cada instancia (fila), y finalmente considera los valores de los *rankings* por algoritmo (columna). Al aplicar este test se han obtenido los valores 2.5 para GENALGO, 2.46 para SS-2P y 1.04 para GRASP_PR con un $p$-valor de $7x10^{-13}$. El ranking medio de nuestro algoritmo es muy cercano a 1, lo que refleja que generalmente obtiene soluciones de mejor calidad que los demás. El $p$-valor tan bajo indica claramente que hay suficiente evidencia estadística para confirmar que hay diferencias entre los tres algoritmos. Posteriormente hemos realizado el test estadístico no paramétrico del signo para detectar las diferencias consistentes entre GRASP_PR y los otros dos métodos. Al comparar GRASP_PR con GENALCO hemos obtenido un valor de 39 con $p$-valor de $3.7x10^{-11}$ y al comparar GRASP_PR con SS-2P también hemos obtenido un valor de 39 con $p$-valor de $1.8x10^{-12}$. Estos valores confirman la superioridad del método propuesto GRASP_PR sobre los otros dos métodos del estado del arte.

## 7 Conclusiones

En este trabajo hemos descrito un algoritmo para obtener soluciones de calidad para el problema SRFLP basado en un GRASP con reencadenamiento de trayectorias (*Path Relinking*). Nuestro método ha sido capaz de obtener soluciones de la misma calidad para las instancias del estado de arte (más pequeñas) que los métodos del estado del arte pero en un tiempo menor. Hemos generado instancias más grandes para comparar los métodos en un escenario más difícil y nuestro método obtiene mejores resultados que los otros métodos en el mismo tiempo de ejecución. La característica más relevante de nuestro algoritmo es su constructivo GRASP, su rápida búsqueda local y un enfoque basado en la distancia de Ulam para el cálculo de las soluciones del reencadenamiento de trayectorias.

Hemos comparado los métodos del estado del arte, tanto con varios conjuntos publicados en el estado del arte como con nuevas instancias más grandes y en todos ellos nuestros método GRASP_PR es competitivo, obteniendo las mismas soluciones pero en menor tiempo (en el caso de las instancias pequeñas) o bien obteniendo mejores resultados en la mayoría de las instancias (en el caso de las instancias grandes). Las instancias presentadas en este trabajo así como las mejores soluciones pueden encontrarse en [14].

## 8 Agradecimientos

Este trabajo ha sido financiado con fondos del Ministerio de Economía y Competitividad a través del proyecto TIN2015-65460-C2-2-P.

## 9 Bibliografía

1. Thomas A. Feo y Mauricio G. C. Resende. A probabilistic heuristic for a computationally difficult set covering problem. *Operations Research Letters*, 8:67—71, 1989.
2. Ravi Kothari and Diptesh Ghosh. The single row facility layout problem: state of the art. *OPSEARCH*, 49(4):442–462, December 2012a.
3. Birgit Keller and Udo Buscher. Single row layout models. *European Journal of Operational Research*, 245(3):629–644, 2015.
4. Ravi Kothari and Diptesh Ghosh. An efficient genetic algorithm for single row facility layout. *Optimization Letters*, 8(2):679–690, February 2014a.
5. Ravi Kothari and Diptesh Ghosh. A scatter search algorithm for the single row facility layout problem. *Journal of Heuristics*, 20(2):125–142, April 2014b.
6. M. G. C. Resende and R. F. Werneck. A hybrid heuristic for the p-median problem. *Journal of Heuristics*, 10(1):59–88, 2004.
7. F. Glover. Tabu search - part 1. ORSA *Journal on Computing*, 1(2):190–206, 1989.
8. F. Glover. Tabu search - part 2. ORSA *Journal on Computing*, 2(1):4–32, 1990.
9. S. M. Ulam. Some ideas and prospects in biomathematics. *Annual review of biophysics and bioengineering*, 1:277–292, 1972.
10. M. Laguna and R. Martí. GRASP and path relinking for 2-layer straight line crossing minimization. *INFORMS Journal on Computing*, 11(1):44–52, 1999.
11. Marc Sevaux and Kenneth Sörensen. Permutation distance measures for memetic algorithms with population managment. In MIC2005, Proceedings of 6th Meta-heuristics International Conference, 832–838, 2005.
12. Miguel F. Anjos and Ginger Yen. Provably near-optimal solutions for very large single-row facility layout problems. *Optimization Methods Software*, 24(4-5):805–817, August 2009.
13. André R. S. Amaral and Adam N. Letchford. A polyhedral approach to the single row facility layout problem. *Mathematical Programming*, 141(1-2):453–477, October 2013.
14. Manuel Rubio-Sánchez, Micael Gallego, Francisco Gortázar, Abraham Duarte. GRASP with path relinking for the single row facility layout problem. *Knowledge Based Systems*. In press.