Desarrollo de Algooritmos Basados en Metaheurísticas para la Resolucióon del Problema de Empaquetado mediante Opt4J

Luís Serrano Hernández, Matías Peirano Maletta

Noviembre 2020

Índice

Introducción

El problema del empaquetado bidimensional, o «SP2D» (del inglés «Strip Packing (2-dimensional) Problem» para abreviar, consiste en tomar una serie de bloques rectangulares y tratar de colocarlos todos de forma que se puedan encerrar dentro de un contenedor lo más pequeño posible. La versión desarrollado en este trabajo modifica el objetivo del problema original y añade un par de restricciones.

Los algoritmos que se presentan en este documento tienen como objeto encontrar una solución partiendo de que el contenedor global ya está establecido y el nuevo objetivo pasa a ser el de almacenar la mayor cantidad de bloques en su interior. En otras palabras, nuestra intención es obtener un algoritmo que, dados una anchura y altura máximaa, así como los rectángulos de los bloques disponibles, sea capaz de colocar la mayor cantidad de bloques dentro del área restringida.

Para el desarrollo de este proyecto se ha intentado diseñar un algoritmo propio, pero, al no dar un resultado aceptable, finalmente se ha optado por consultar la red en busca de literatura relativa al problema en cuestión. Así pues, el artículo raíz del que brotan las ideas principales de este trabajo es el publicado por Mitsutoshi et al. [1].

Descripción del Método Aplicado

Del artículo mencionado anteriormente se obtiene una buena toma de contacto con el problema, así como una serie de posibles enfoques. De entre ellos se han escogido dos, el redactado por Hiroshi et al. [2], un desarrollo de ingenieros de IEEE en el año 1996, y el propuesto por Zhang et al. [3]. El primero es muy interesante, pues ofrece una visión muy práctica de las aplicaciones que puede tener el algoritmo, la disposición de circuitos eléctricos es la que recibe el enfoque en este caso. No obstante, se ha considerado que su complejidad excede el interés de este trabajo, por lo cual se ha optado por implementar la segunda de las propuestas.

La versión implementada no es exactamente la que se describe en [3], pero ha sido la principal inspiración. En un principio, esta propuesta estaba enfocada a su uso en combinación con un algoritmo genético, no obstante, con alguna ligera modificación, se ha empleado también para el algoritmo de enfriamiento simulado. Son, por tanto, dos los algoritmos basados en metaheurísticas los implementados en este trabajo, un algoritmo genético, o «EA» (del inglés «Evolutionary Algorithm») y un algoritmo de enfriamiento simulado, o «SA» (del inglés «Simulated Annealing»). Tanto el genotipo, como el fenotipo y también el algoritmo de evaluación son idénticos para ambos algoritimos (aunque en «SA», al no tener nada que ver con la genética, los términos son más bien la forma interna y externa de representar la solución). A continuación se describen brevemente.

Genotipo

El genotipo representa como es internamente el individuo. Este debe ser compacto, para no implicar una gran carga computacional, pero al mismo tiempo debe de permitir explorar la mayor cantidad de soluciones posibles. Hay que tener en cuenta que el diseño también es importante de cara a diseñar los algoritmos de cruce y mutación. Teniendo en cuenta todo lo anterior, y con la mente puesta en el algoritmo propuesto por Zang et al. [3], se plantea representar al individuo como una permutación de los bloques que están disponibles para ser colocados.

Para mantener las cosas simples, cada bloque es representado por un identificador único, que puede ser, por ejemplo, la posición que ocupa en la lista de bloques. A primera vista podría parecer que es muy complicado asegurar si un individuo es válido, pues no hay que olvidar que hay restricciones muy fuertes, pero la clave para este dilema se encuentra en la forma en la que se decodifica el genotipo para conseguir el fenotipo, de lo cual se habla en la siguiente subsección.

Fenotipo

El fenotipo es la versión externa del individuo, podemos decir que es la solución en si misma. En esta parte toca subsanar el desafío de convertir la lista de bloques permutados en una representación fácil de interpretar.

Tal y como se ha ido anticipando, es el turno de aplicar lo aprendido en [3], la versión del algoritmo haría lo siguiente: tomaría los bloques de uno en uno, según su disposición en el genotipo, entonces para cada bloque trataría de colocarlo pegado a la esquina inferior derecha del bloque, de manera que se van colocando uno al lado del otro horizontalmente hasta que uno no cabe, seguidamente se empiezan a colocar con un desplazamiento en la altura igual a la altura del mayor bloque colocado en la fila recién construida, y para acabar este proceso se repite hasta que se alcanza el límite vertical, a partir de este punto los bloques ya no forman parte de la solución.

Para el diseño del fenotipo se ha optado por una lista de coordenadas del mismo tamaño que la que contiene los bloques a colocar. Las coordenadas del fenotipo representan entonces la posición del bloque en la misma casilla de la lista de tamaños de bloques en la solución. Las coordenadas del fenotipo se inicializan a un valor que no pueda ser tomado en la solución, de esta manera se distinguen fácilmente aquellos que no se han podido colocar de los que sí. Esto, como se va a explicar en breve, hace más sencilla la implementación de un evaluador.

Evaluación

La evaluación es también una parte crucial, en ella se toma el fenotipo decodificado y se le atribuye un valor. Teniendo en cuenta que nuestro objetivo

desde el principio ha sido obtener la función que maximice el número de bloques colocados, esta evaluación tiene que premiar correctamente a los individuos.

En este caso es tan sencillo como recorrer el fenotipo en busca de aquellas coordenadas que si pueden existir dentro del entorno del problema y contabilizar cuantas hay. Es, en resumidas cuentas, como contar el número de bloques que el decodificador a marcado como pertenecientes a la solución y a los que les ha asignado una representación en su mundo.

El uso de una evaluación más compleja, como el caso de emplear funciones multiobjetivo, ha sido descartada, pues se ha considerado que, si bien podría ser interesante enfrentar el hecho de ocupar el menor espacio posible con colocar el mayor número de bloques podría ser interesante, escapa a los objetivos planteados inicialmente.

Ejemplo

Dado que lo expuesto en esta sección puede aparentar abstracto de primeras, se ha decidido realizar un pequeño ejemplo gráfico que servirá para despejar las principales dudas. El ejemplo lo componen las figuras 1, 2 y 3 que representan los bloques junto al contenedor, una mala solución al problema y una solución óptima al problema respectivamente. En la figura 1 los bloques disponibles están representados por rectángulos grises que tienen contenido su identificador. El área blanca representa el tamaño del contenedor y la unidad de tamaño mínima es equivalente al valor de los lados de los bloques 16 y 20. Las otras dos figuras comparten esquema de representación entre sí, pero difieren significativamente de la primera.

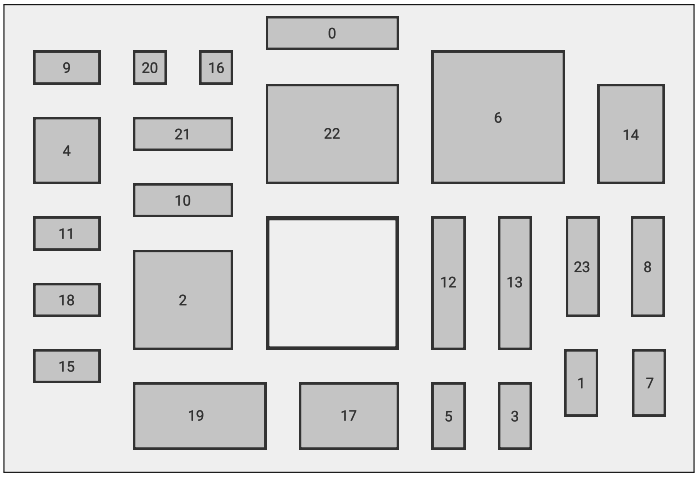

Figura 1: Ejemplo de instancia del problema: Bloques y contenedor

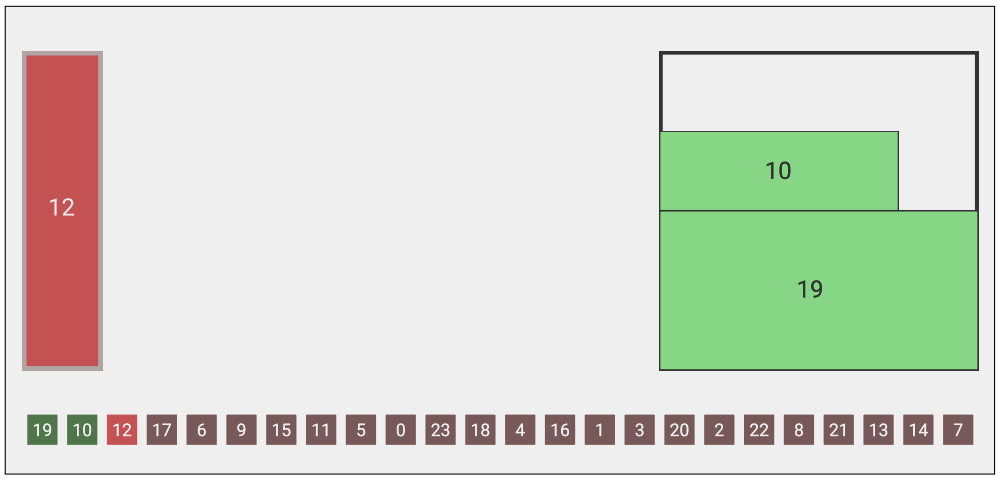

Figura 2: Ejemplo de instancia del problema: Una mala solución

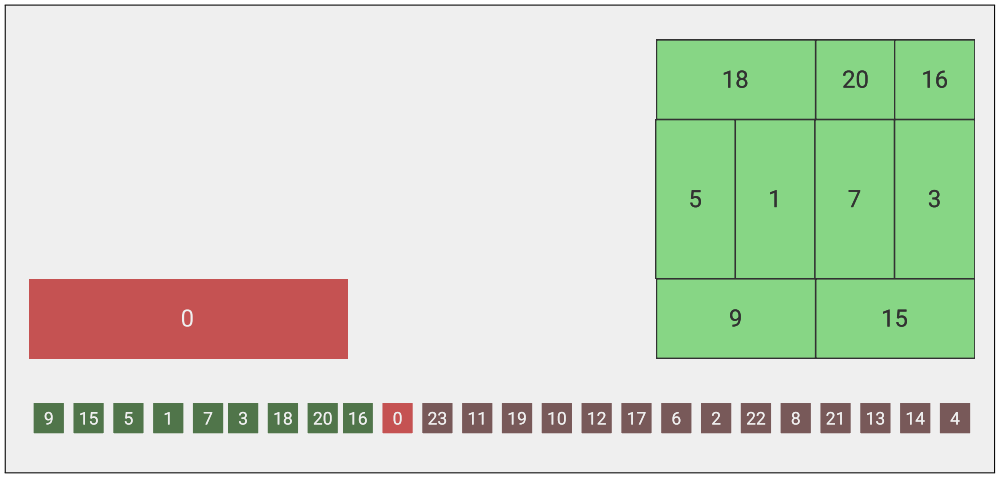

Figura 3: Ejemplo de instancia del problema: Una solución óptima

La cadena de bloques que se puede apreciar en la parte inferior de las figuras 2 y 3 representa la forma implícita de la solución, en el caso del algoritmo genético es el genotipo. En esta ocasión el fenotipo es una representación gráfica que emplea distintos tipos de rectángulos. Se ha decidido hacerlo así por ser más visual, el fenotipo evaluable tendría la siguiente forma para el caso de la solución de la figura 2: «[(-1, -1), (-1, -1), (-1, -1), (-1, -1), (-1, -1), (-1, -1), (-1, -1), (-1, -1), (-1, -1), (-1, -1), **(0, 2)**, (-1, -1), (-1, -1), (-1, -1), (-1, -1), (-1, -1), (-1, -1), (-1, -1), (-1, -1), (-1, -1), **(0, 0)**, (-1, -1), (-1, -1), (-1, -1), (-1, -1)]]». Esta sería una mala solución, pues si contamos las coordenadas que no han variado su valor

inicial tenemos que solo hay dos bloques colocados. Por otro lado, la solución externa representada en la figura 3 tiene el siguiente aspecto: «[(-1, -1), **(1, 1)**, (-1, -1), **(1, 3)**, (-1, -1), **(0, 1)**, (-1, -1), **(2, 1)**, (-1, -1), **(0, 0)**, (-1, -1), (-1, -1), (-1, -1), (-1, -1), (-1, -1), **(2, 0)**, **(3, 3)**, (-1, -1), **(0, 3)**, (-1, -1), **(2, 3)**, (-1, -1), (-1, -1), (-1, -1)]]». La suma de bloques colocados asciende a 9, la cual es, de hecho, una de las múltiples soluciones óptimas al problema. El fácil ver que variaciones tan simples como barajar las posiciones de los bloques 1, 3, 5 y 7 entre sí en el genotipo produce otras soluciones también óptimas. Incluso para un problema tan pequeño las posibilidades son muy grandes. Esto último no hace del problema algo más sencillo, ni mucho menos, pero eso se obervará mejor más adelante. Ahora es el momento de ponerse manos a la obra y de pasar de la teoría a la práctica.

Implementación de la Solución

Como dice el título del trabajo, la solución ha sido implementada mediante Opt4J, un marco de desarrollo enfocado a la implementación de metaheurísticas. Opt4J es también empleable como una biblioteca y es, de hecho, el mayor uso que se le ha otorgado. La implementación completa es muy extensa, por lo que se van a tratar solo los puntos principales.

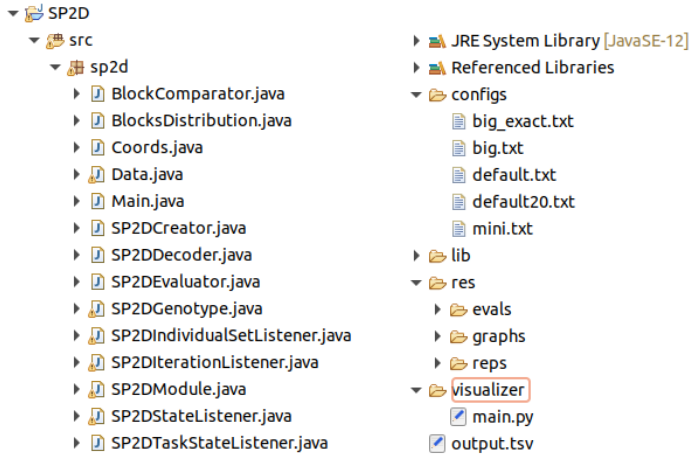

Figura 4: Vista de la Estructura (reducida) del proyecto en el IDE Eclipse

En la figura 4 se puede observar la estructura final del proyecto en Eclipse. Empecemos por el arranque, la clase «Main» contiene un método ejecutable, implementado en base a lo explicado en los tutoriales de [4], de esta forma las ejecuciones son más fáciles de manejar que empleando el lanzador propio de Opt4J. «Data» es una extensa clase en la que se encuentran todos los parámetros de configuración, así como una serie de variables y métodos auxiliares a los que se recurre desde el resto del proyecto.

Siguiendo con la explicación, las clases cuyo nombre comienza con «SP2D» son implementaciones de las interfaces de módulos de la biblioteca de Opt4J, a nivel básico solo se requieren cuatro («Creator», «Decoder», «Evaluator» y «ProblemModule»), pero de esta forma se ha logrado un mayor nivel de personalización. Fijando la vista en el paquete del directorio «src» se puede ver que uno de estos módulos implementados es el relativo al genotipo, esto se ha hecho así para poder hacer uso de unos métodos de inicialización más complejos. «SP2DGenotype» es una especialización del genotipo de permutación, una buena elección teniendo en cuenta lo previamente explicado. Los «listeners» se han implementado con la intención de facilitar la monitorización de las ejecuciones.

Otro propósito que se ha extraído de estos es el de lograr imprimir a dos archivos los resultados de cada ejecución. El primero es un «TSV», es decir, una serie de líneas de texto con valores separados por tabulaciones, ideal para las representaciones gráficas, que contiene los resultados de monitorizar las iteraciones. El segundo contiene código en «JSON» con el que se representa, tanto el problema, como las distintas mejores soluciones que se han ido encontrando a medida que avanzaba el algoritmo. El hecho de ser «JSON» hace muy sencilla su portabilidad, lo que se ha aprovechado para crear un programa que permite visualizar las soluciones en el lenguaje de programación «Python», puede encontrrse en el directorio «visualizer». Todos estos archivos son almacenados en el directorio «res», que se divide en tres subdirectorios, «evals» almacena los «TSV», «reps» alamcena los «JSON» y «graphs» almacena los «PDF» generados por el programa principal del «visualizer».

Las últimas clases restantes se han empleado para hacer más entendible el código, son un apoyo que hace más sencillo el mantenimiento del proyecto. «BlockComparator», por ejemplo, se emplea para sacar mejor provecho a las bibliotecas de «Java» haciendo uso de sus algoritmos de ordenación. Lo cual enlaza con otro punto importante.

En cuanto a las posibles configuraciones, se ha añadido la opción de ordenar los bloques a colocar antes de inicializar el problema. Esta es una recomendación que se hace en [3], pues finalmente, también se ha aplicado el algoritmo, no solo para la decodificación, sino para la creación. En realidad, la versión original del algoritmo estaba enfocada a la creación de los genotipos, por lo que se ha considerado que la mejor opción era, en efecto, implementar una versión adaptada en el genotipo de propia creación. Por otro lado, se han creado una serie de ficheros de configuración para precargar instancias del problema, la de nombre «default» es la instancia propuesta en el trabajo original, la cual se ha limitado a un área de dieciséis por dieciséis unidades de tamaño. Estos ficheros de configuración todavía no han sido traducidos a «JSON», pues esa fue una decisión posterior. El resto de valores que se pueden configurar normalmente en Opt4J también están disponibles para personalizar, tanto los de «EA», como los de «SA», en el siguiente punto se presenta los resultados obtenidos.

Evaluación

Esta sección es la encargada de demostrar que el duro trabajo a merecido la pena, es hora de presentar los resultados. No obstante, antes de correr demasiado, es importante determinar que los algoritmos funcionan a un nivel básico, es decir, que no devuelven soluciones no factibles y que consiguen mejorar las soluciones con el tiempo de computo. Para ello se ha creado una instancia muy pequeña del problema, de nombre «MINI» y cuya configuración puede consultarse en el repositorio que almacena el proyecto [7] o en el ejemplo de la sección en la que se describe el diseño aplicado, pues es la misma. Los datos mostrados han sido creados automáticamente partiendo los de «TSV», con quinientas muestras por dato, y «JSON» mediante el visualizador realizado en «Python», implemetado haciendo uso de la biblioteca «matplotlib», cuya documentación [5] ha sido realmente útil. Vamos a aprovechar las mencionadas comprobaciones para explicar como están estructurados.

Los resultados de las pruebas pueden verse en las figuras 5 y 6. En este caso, cada figura contiene un par de soluciones junto a sus representaciones. En cada grupo de estas encontramos cuatro gráficos, independientemente del algoritmo. El cual, por cierto, está indicado como título de cada grupo. En el gráfico superior izquierdo está representada la evolución de la solución con respecto al número de evaluaciones, dependiendo del tipo de algoritmo otos datos relevantes estarán también representados. En cuanto al superior derecho, tenemos el más simple del conjunto, es una representación que muestra la acumulación del tiempo empleado para el cómputo a medida que aumentan las evaluaciones. Y por último, los dos inferiores tienen la misma estructura de representación, el izquierdo muestra la configuración de la solución inicial y el derecho de la final, ambas obtenidas partiendo del fenotipo antes expresado. Hay que tener en cuenta que las gráficas ocupan aproximadamente el mismo volumen, así que, soluciones con contenedores de lados muy dispares pueden tener representaciones deformadas en algún eje. En realidad, aún quedan unos detalles, que desde luego, son importantes.

Bajo cada tupla de gráficas de ambas figuras se presentan una serie de atributos, estos representan los argumentos de las ejecuciones para los resultados obtenidos. La gran mayoría son fáciles de interpretar, pues representan valores característicos de sus respectivos algoritmos, pero para un par es algo distinto. «S.F.» indica si los bloques han sido ordenados previa ejecución del algoritmo. Pasando al siguiente, «Case» hace referencia al tipo de instancia sobre la que se ha ejecutado, donde «MINI», «DEFAULT» y «BIG» son algunas de las instancias predefinidas, hay alguna especial. En concreto, «DEFAULT» es la instancia correspondiente a la propuesta original del trabajo a la que, a falta de nociones que indicasen las restricciones de tamaño, se ha establecido que el contenedor sea de dieciséis unidades de ancho por dieciséis de largo. Otro caso especial es el de «RANDOM», en el cual, tras este, se indican el número de bloques de la instancia, los límites establecidos y el porcentaje en proporción del lado más grande para un bloque en función del lado respectivo del contenedor. Con esta información podemos pasar a ojear las evaluaciones.

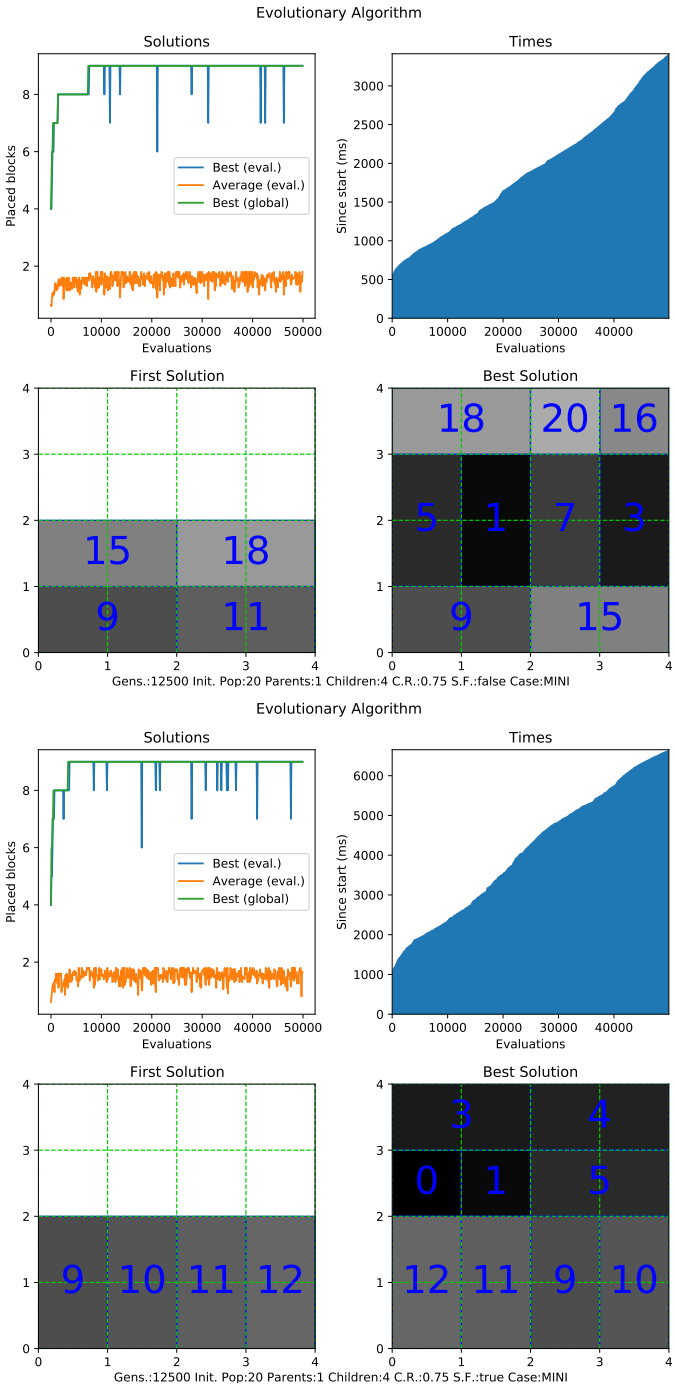

Figura 5: Resultados tras ejecutar el algoritmo genético sobre el caso «MINI»

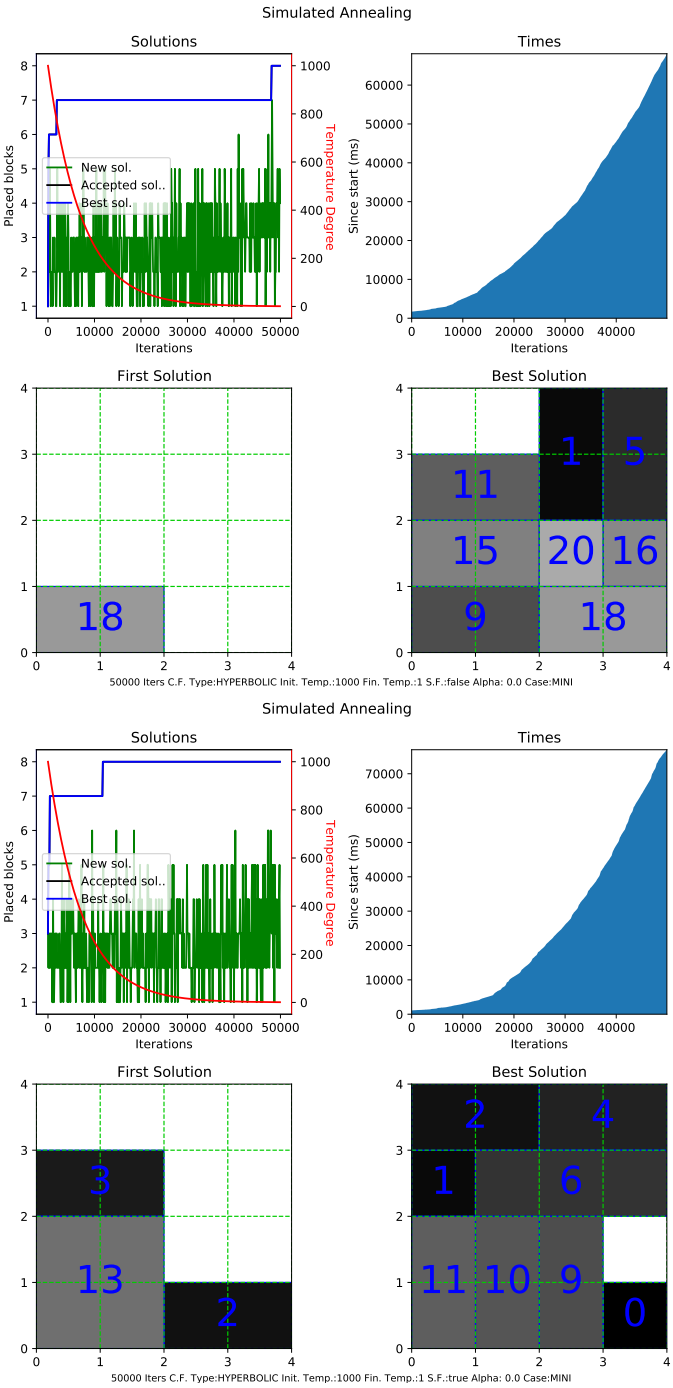

Figura 6: Resultados tras ejecutar el algoritmo de enfriamiento simulado sobre el caso «MINI»

Como se puede observar, tanto en la figura 5 como en la 6, ambos algoritmos mejoran las soluciones de las que parten con el tiempo, hasta que convergen, al menos. La factibilidad está asegurada también, pues es el decodificador el que determina la estructura del fenotipo y en el momento que un bloque se sobresale por el eje de las ordenadas la solución se da por completada. En el caso del «SA» parece que la solución mejore en ocasiones sin corresponderse con la solución mostrada para la iteración, pero esto es debido a que, de las, en este caso, cincuenta mil muestras de información, se toman quinientas en total, provocando un salteo importante. Se hace así para aligerar el proceso de creación de la gráfica, reducir su tamaño y evitar borrones provocando, eso sí, esas anomalías en la representación.

Otra oportunidad que brinda esta instancia es que, al haber sido manualmente preparada, se ha pretendido conocer previamente la mejor solución. El valor de esta es el de nueve bloques colocados dentro del área cuadrada de lado cuatro, sin embargo, no es una solución única, ni siquiera en el tipo de bloques necesarios para su composición. Esto es perceptible en los resultados aportados sobre el «EA», es decir, los de la figura 5. La otra figura, la 6 representa los resultados sobre el «SA». En ambos se aprecia, pero en este último más, que cada iteración ofrece valores que pueden ser muy variantes con respecto a la anterior. Hay que tener en cuenta que los identificadores de los bloques en las gráficas de los resultados los establece el generador en función de la posición del bloque al que representan en la lista de tamaños de bloques, por lo que para una instancia del problema hay dos posibles conjuntos de identificadores, los que representan la versión ordenada y los que no. Con todo lo dicho y con la retrospectiva de la sección en la que se habla del modelo de la solución, podemos decir que no hay ningún comportamiento extraño aparente.

Algoritmo Genético

La realidad es que hay un dato especialmente preocupante que se puede extraer de las pruebas anteriores: el coste temporal de la ejecución del algoritmo es cada vez mayor a medida que la ejecución avanza. Lo cual parece más acusado incluso en el caso del «SA». Basándonos en la teoría y observando el código fuente [6] son comprensibles unos mayores costos temporales para el arranque del «EA» o cuando «SA» está todavía aceptando muchas soluciones, pero el comportamiento descrito es ciertamente inusual. Los dos motivos posibles que se consideran son un uso excesivo de memoria, por la forma en la que se realizada el monitorizado y la extracción de datos, causante de demasiados accesos a la zona «swap» del disco o un posible sobrecalentamiento de la CPU. Esto último no se descarta debido a que la máquina en la que se han realizado las ejecuciones posee un ventilador antiguo que se activa en muchas ocasiones, incluso sin realizar un uso exigente del procesador.

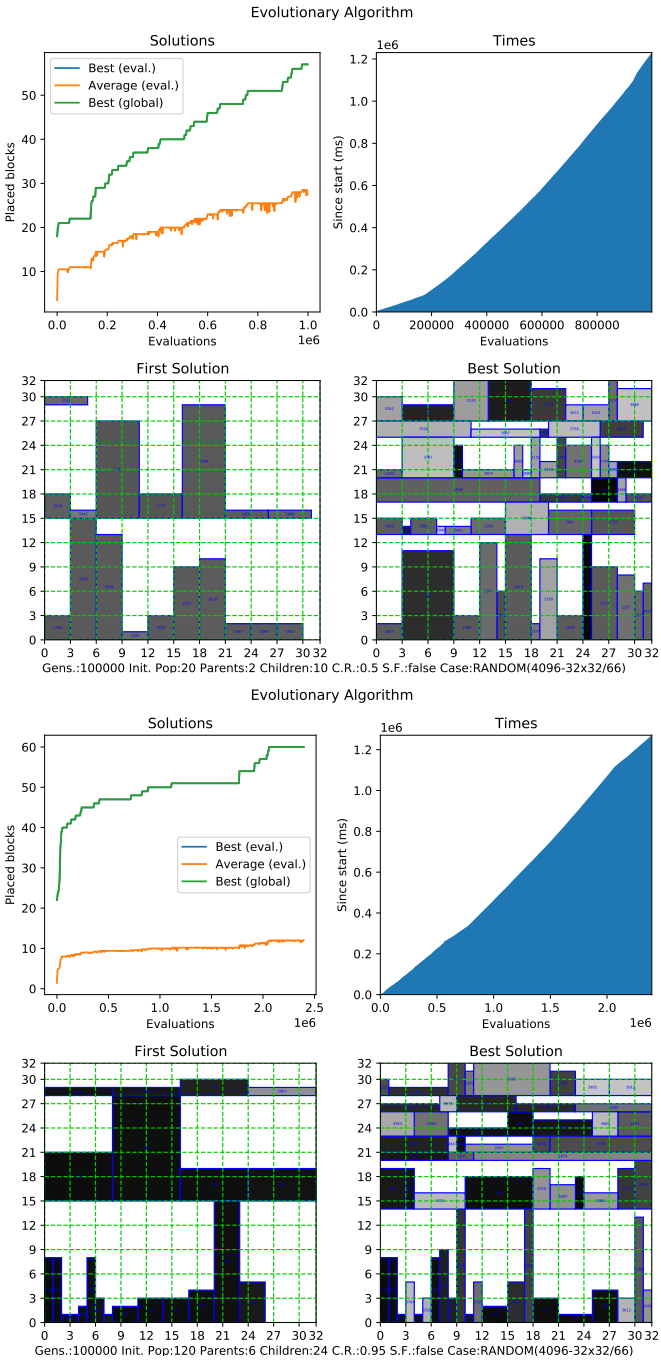

Figura 7: Resultados tras ejecutar «EA» sobre un caso «RANDOM» de contenedor de 32x32 uds., 4096 bloques de hasta un 66 % de su tamaño y sin ordenar.

El asunto es que, sea cual sea el motivo, al no haberlo podido solucionar, hay que plantear bien la estructura de trabajo. Antes de lanzar ejecuciones de muchas generaciones para el caso «DEFAULT», se han realizado una serie de pruebas de más corta duración sobre instancias aleatorias de mayor tamaño de contenedor y número de bloques. La intención es encontrar unos buenos parámetros para la ejecución definitiva. Se ha establecido que el tamaño de los bloques pueda ser como mínimo de unidad por lado y de máximo dos tercios respecto al lado del contenedor para cada eje correspondiente, En descripción es similar al problema original, pero con quinientas doce veces más bloques, para incrementar el número de soluciones posibles

Es la figura 7 la que contiene los resultados más interesantes de estas ejecuciones. Representan dos estrategias bastante distintas, claramente ambas con el «EA», pero con parámetros de distinto orden. Antes de hablar de los datos, observar que la unidad en la que se comparan ambas ejecuciones es el tiempo, pues ambas tardan más o menos lo mismo, aunque difieran en casi todo lo demás (excepto las generaciones, que son las mismas también). La primera ejecución cuenta con 20 individuos en la primera generación, en cada generación un 10 % de los mejores individuos, ya que la selección es elitista, generará al 50 % de la siguiente generación con un ratio de cruce del 50 %. Por otro lado, la segunda ejecución parte de una generación inicial de 120 individuos, de los cuales solo el 5 % será padre y contribuirá a reponer el 20 % de los individuos para la próxima generación, con un ratio de cruce del 95 %.

La solución inicial para ambos es bastante similar, pero para el momento en el que los algoritmos se paran, la segunda ejecución obtiene una solución casi un 10 % superior a la de la primera. No obstante, si nos basamos en las evaluaciones, la segunda ejecución en la evaluación diez millones, que es cuando se detiene la primera, tiene una solución inferior a esta. Llama la atención la diferencia en términos de computación, pues la segunda, en el mismo tiempo, hace más del doble de evaluaciones que la primera, se considera poco fiable la estimación temporal, debido a lo anteriormente expuesto, pero una diferencia tan grande es difícil de excusar. También, el arranque de la segunda ejecución es mejor, pero se desinfla rápidamente mientras que la otra parece mantener una tendencia creciente durante toda la ejecución, de haber seguido seguramente la hubiese alcanzado, sobretodo teniendo en cuenta la solución media por generación, que aunque en la segunda permanezca casi estática, la primera cada vez tiene mejores generaciones, hasta el punto de que el valor de la solución media en la última generación es mayor a la mejor solución de la primera.

Basándonos en las decenas de ejecuciones y en lo recién reflexionado, lanzamos dos ejecuciones a un millón de generaciones cada una. Una repetirá los parámetros de la primera anterior, mientras que la otra será una alternativa bastante diferente, más influenciada por los otros resultados. Los resultados son los mostrados en la figura 8. La segunda ejecución realiza cuatro veces más evaluaciones, pero no parece algo relevante, ya que ambos algoritmos convergen enseguida. Aún así, de nuevo, sorprendentemente es la primera ejecución la que más tiempo se demora en realizar los cálculos. El valor de la solución media se mantiene entre seis y ocho en ambos casos durante toda la ejecución, con

algo más de varianza en el caso de la primera ejecución, seguramente debido al bajo ratio de cruce. El valor de solución óptima alcanzado por ambos es el mismo, diecisiete bloques colocados, eso sí, pese a que parten de la misma solución inicial, las soluciones finales difieren bastante en composición. A simple vista y sobre el hueco que queda en cada representación, el resultado de la primera aparenta más vacío. Con un poco más de atención se puede ver que en la primera el patrón es el de dos filas exteriores de rectángulos de mayor altura que anchura y una central de más anchura que altura, mientras que en el caso de la segunda es justo al revés. Ambas soluciones incluyen los dos bloques más pequeños (10 y 18). Sin resolver el problema por cualquier otro medio, diría que podemos estar ante la solución óptima, pero también podría no ser el caso. Echando un ojo a las representaciones se puede ver que aprovechan bastante bien los espacios.

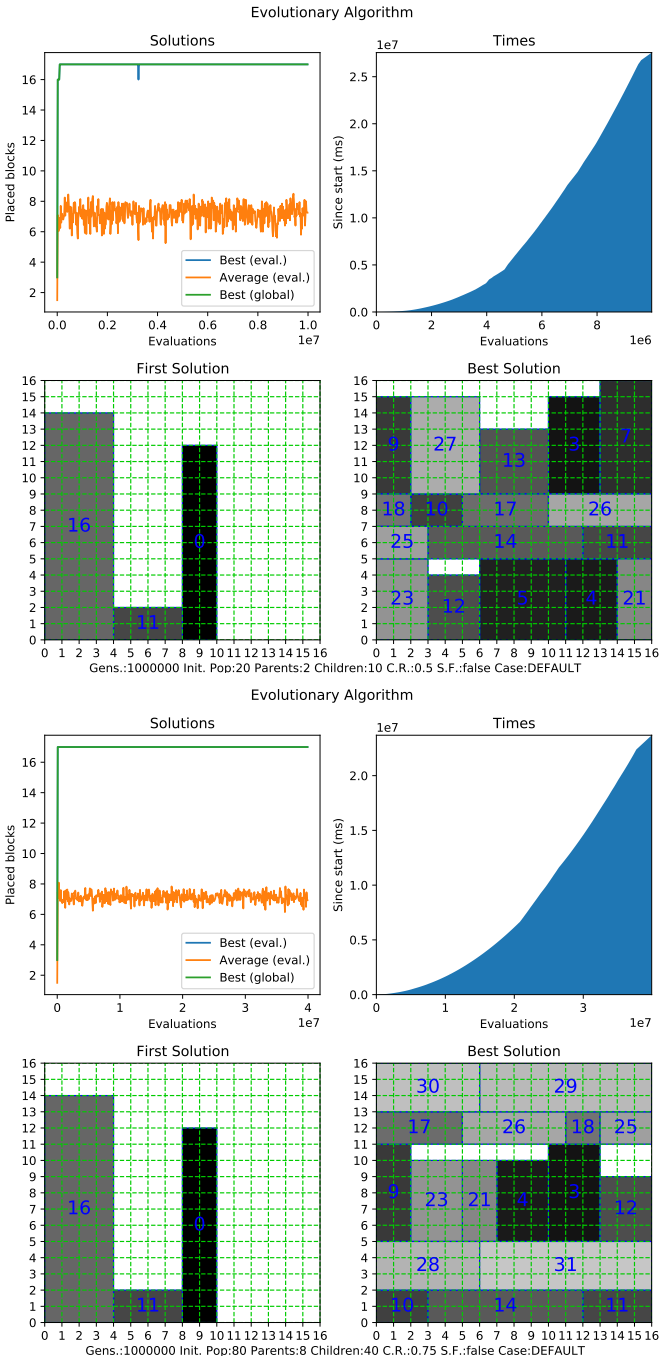

Figura 8: Resultados tras ejecutar «EA» sobre el caso «DEFAULT».

Enfriamiento Simulado

Las mismas ideas que se han explicado en la subsección anterior se pueden aplicar aquí, por lo que lo primero va a ser realizar diversas ejecuciones aleatorias con los mismos parámetros que antes, se seleccionarán las mejores y se replicarán con más duración para el caso que nos ocupa. En el caso del «EA» se ha evitado a conciencia comenzar con un grupo preordenado de bloques, pues sería una forma muy directa de cortar la exploración, que ya de por si es limitada dada la versión implementada por defecto del algoritmo en el entorno de desarrollo. En el caso del «SA» ha sido al contrario, se ha considerado que su convergencia era tan lenta que es necesario hacer uso de la heurística de Zhang et al. [3]. Hablemos de resultados, que se entenderá mejor. Algunas de las muchas pruebas realizadas se pueden ver tanto en la figura 9 como en la 10.

En ambas figuras están las evoluciones de la mejor solución con el paso de las iteraciones para distintos métodos de enfriamiento, la línea roja representa la temperatura. En el caso de la figura 9, esta representa ejecuciones sin hacer uso de la heurística de preorednación, mientras que la figura 10 representa ejecuciones que si la aplicaron. Es fácil distinguir entre los tipos de enfriamiento empleados, la primera de la figura 9 y las tres primeras de la figura 10 reducían la temperatura exponencialmente, la segunda de la 9 y cuarta, quinta y sexta de la 10 aplicaron una reducción hiperbólica y el resto aplicaron un decremento lineal de la temperatura. De cualquier forma, los resultados son bastante descorazonadores, sea la combinación de parámetros que sea, todos parecen dar un pésimo resultado.

Para comparar, tenemos el «EA» que acabamos de analizar. No sería justo basar la comparación en el número de generaciones, al igual que tampoco lo hemos hecho antes, pues dependiendo del número de hijos del genético pueden variar mucho, por lo que una opción es interpretar los resultados en base a las llamadas al evaluador.

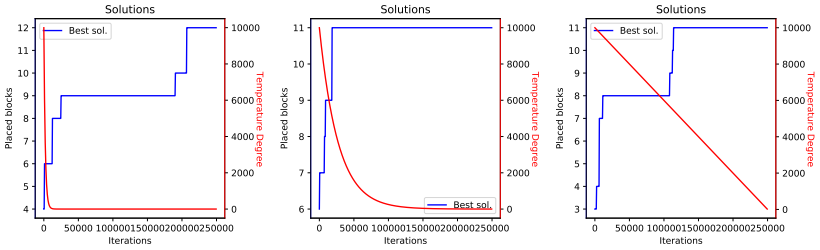

Figura 9: Resultados tras ejecutar «SA» sobre un caso «RANDOM» de contenedor de 32x32 uds., 4096 bloques de hasta un 66 % de su tamaño y sin ordenar.

En las doscientas cincuenta mil evaluaciones que se han realizado para estas pruebas de «SA» se ha obtenido una alta tasa de invariabilidad, generando una enorme dependencia hacia la solución inicial, siendo el mejor de los casos, uno en el que la diferencia entre el valor de la solución inicial y la final distaban

ocho unidades. Son resultados bastante pobres, teniendo en cuenta que con esta cantidad de evaluaciones el genético obtiene de media soluciones hasta un 50 % mejores.

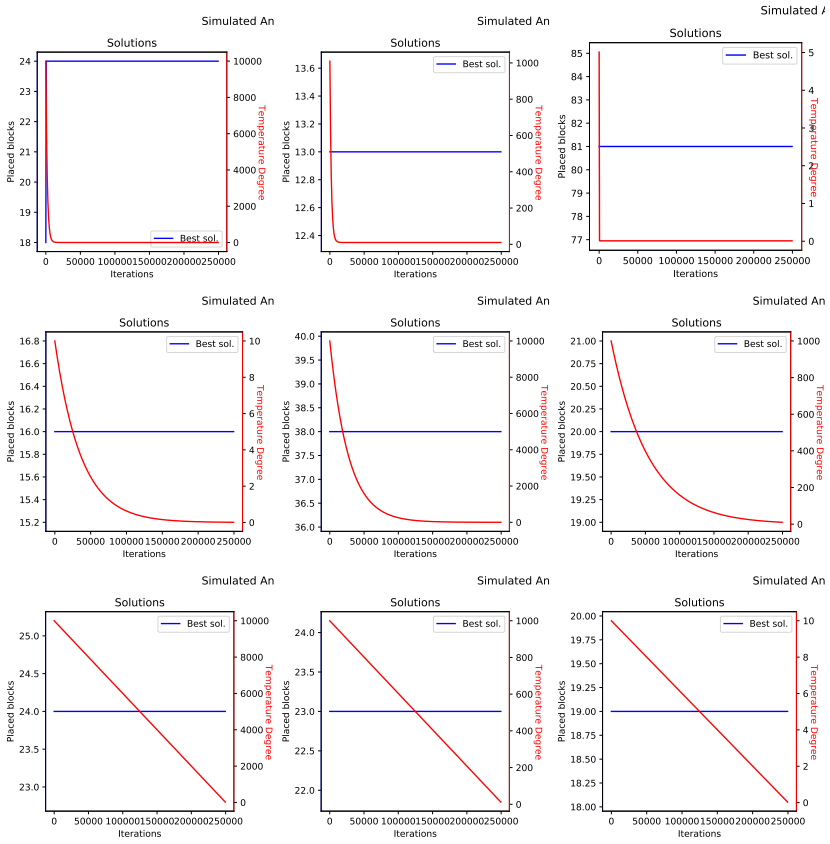

Figura 10: Resultados tras ejecutar «SA» sobre un caso «RANDOM» de contenedor de 32x32 uds., 4096 bloques de hasta un 66 % de su tamaño y ordenanados.

Es importante conocer porque se da este suceso, así que se han realizado algunas ejecuciones más, monitorizando una mayor cantidad de datos. Los resultados en cuestión se muestran en la figura 11. En adición a los datos de los resultados anteriores, en estas ejecuciones se pueden observar también las nuevas soluciones que aparecen, obtenidas del vecindario de la anterior guardada, así como el valor acumulado de nuevas soluciones guardadas. Este último dato se ha trasladado a la gráfica de los tiempos para no saturar mucho la primera. Independientemente del esquema de enfriamiento las ejecuciones presentan patrones similares. Las nuevas soluciones tienden a mantenerse dentro de un intervalo, el coste de ejecución crece exponencialmente todas las adiciones se realizan al comienzo. Se ha llegado a considerar que se trate de un error de

configuración, pero tras revisar innumerables veces el código, se ha optado por realizar la ejecución con los datos que conocemos, pues no se ha encontrado respuesta.

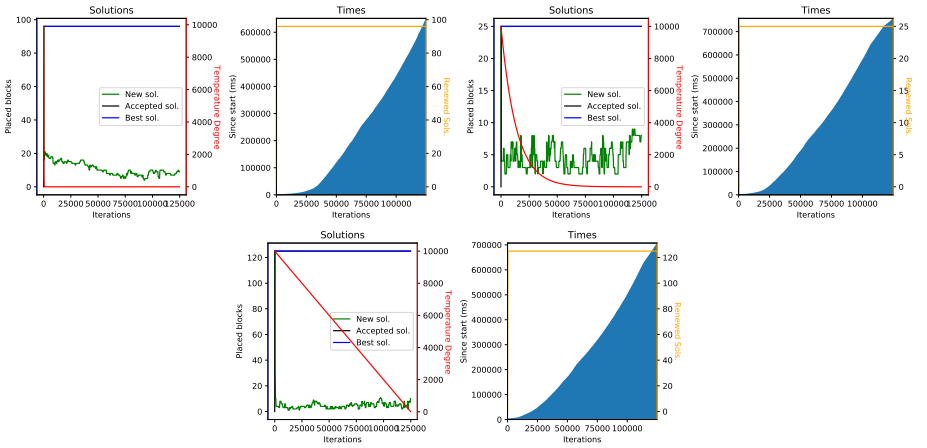

Figura 11: Resultados tras ejecutar «SA» sobre un caso «RANDOM» de contenedor de 32x32 uds., 4096 bloques de hasta un 66 % de su tamaño y ordenados (más detalle).

En base a todas las ejecuciones, la opción de emplear un decremento lineal se descartó, por lo que se procedió a lanzar cuatro ejecuciones, dos con esquema exponencial y dos con esquema hiperbólico. También se jugó con la posibilidad de desactivar la heurística de ordenación, ya que, al ser tan pocos los bloques, no daba los mismo resultados que en el caso «RANDOM» que tenía muchos más, ese fue un fallo importante de extrapolacion. Tras esto, los dos resultados más relevantes son los presentados en la figura 12.

Los primeros, en la parte superior, corresponden a una ejecución del algoritmo con un esquema de decremento exponencial y sin realizar una previa ordenación de los bloques, los resultados de la parte inferior representan a la segunda ejecución, la cual aplicaba un esquema de decremento hiperbólico, también sin ordenación previa. Viendo las soluciones iniciales y finales, se nota la gran diferencia en el caso de la primera ejecución, en ella solo se conserva el bloque 10 con respecto a la inicial. Mientras tanto, en la segunda ejecución son varios los bloques que se encuentran en ambas, además su valor no varía mucho, seguramente serían vecinos no muy lejanos. La primera ejecución parte de una solución más baja, pero dispone de más espacio libre, mientras que sucede al contrario en el caso de la segunda ejecución. Finalmente es la primera la que obtiene el mejor resultado de ambas, pero esta vez es fácil asegurar que no se trata de la solución óptima, pues hemos obtenido soluciones mejores con los algoritmos genéticos.

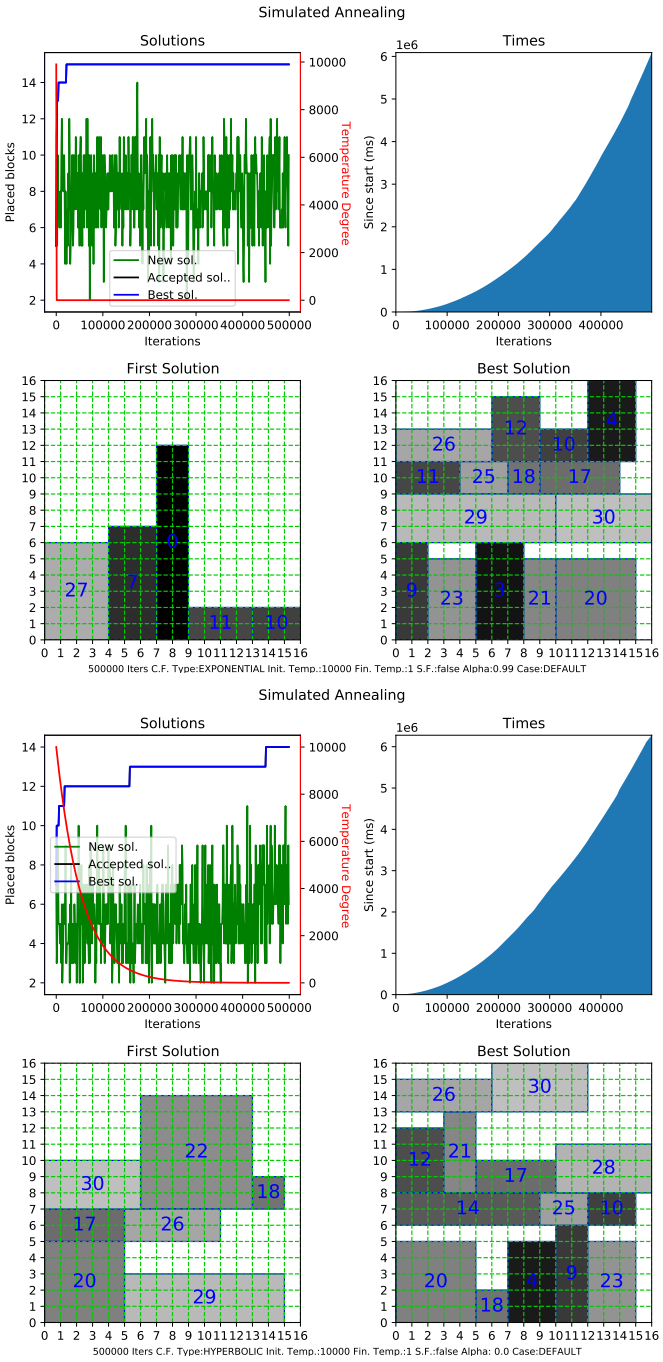

Figura 12: Resultados tras ejecutar «SA» sobre el caso «DEFAULT».

Conclusiones

Han sido muchas las cosas aprendidas durante el desarrollo de este proyecto, la memoria a resultado extensa, pero si incluyese todo lo dedicado al proyecto, bien podría triplicar su extensión. Esa ha sido una de las claves del proyecto, la falta de concisión y la dificultad para enfocar bien la meta y las planificaciones han hecho de este trabajo una tarea más ardua de lo que seguramente debería haber sido.

En cuanto a los resultados obtenidos, creo que podemos estar contentos con el rendimiento del algoritmo genético, ya que obtiene buenos resultados y converge a gran velocidad. Por el contrario, el algoritmo de enfriamiento simulado ha quedado bastante peor parado. Es comprensible, pues el artículo en el que está basada la heurística empleada para la creación y decodificación de los individuos planteaba los algoritmos genéticos como la herramienta para la ocasión. No obstante, la representación del genotipo considero que es perfectamente válida para ambos casos, quizás este tipo de problemas no sean los correctos para aplicar el «SA». De todas maneras, de haber sido capaz de comprender la biblioteca completamente se habría tratado de implementar una versión del algoritmo con recalentamiento, que parecía ser la ficha restante. El otro de los grandes trabajos pendientes ha terminado siendo la capacidad para retomar una ejecución donde termino otra, pues, si bien es cierto que ya se guarda la solución en «JSON», no es posible cargarla. El mayor muro de cara a trabajar con Opt4J como biblioteca ha sido tener que enfrentarse a «GUICE» y los inyectores, que es la forma en la que los desarrolladores establecieron las comunicaciones entre los distintos módulos, nunca me había enfrentado ha esta biblioteca y sigo muy lejos de conocerla bien.

En conclusión, ha sido un trabajo al que se le han dedicado cientos de horas, pero la mala planificación o enfoque han lastrado parte de su desarrollo, no hay que olvidar que la mitad del tiempo se dedicó a desarrollar una versión en C++ que ni se ha mencionado. Aún con todo, se ha realizado un trabajo exhaustivo, se ha exprimido todo lo que se ha podido de la biblioteca y se han implementado unos buenos algoritmos, el balance es muy positivo. Con unos días más de trabajo se debería reorganizar el código, tratar de optimizar algunas partes e implementar las dos carencias principales anteriormente mencionadas. El trabajo ha sido fructifero.

Referencias

[1] Mitsutoshi Kenmochi, Takashi Imamichi, Koji Nonobe, Mutsunori Yagiura, Hiroshi Nagamochi Exact algorithms for the two-dimensional strip packing problem with and without rotations. *European Journal of Operational Research*, 198:73–83, 2009.

[2] Hiroshi Murata, Kunihiro Fujiyoshi, Shigetoshi Nakatake, Student, and Yoji Kajitani VLSI Module Placement Based on Rectangle-Packing by the

Sequence-Pair. *IEEE Transactions on Computer-Aided Design of Integrated Circuit and Systems*, 15-12, Diciembre 1996

[3] Zhang De-Fu, Chen Sheng, Yan-Juan An Improved Heuristic Recursive Strategy Based on Genetic Algorithm for the Strip Rectangular Packing Problem. *Acta Automatica Sinica*, 33-9, Septiembre 2007

[4] Opt4J: A Modular Framework for Meta-heuristic Optimization - Documenation. *Consultado en* `http://opt4j.sourceforge.net/documentation.html`

[5] Overview - matplotlib 3.3.3 documentation. *Consultado en* `https://matplotlib.org/3.3.3/contents.html`

[6] Opt4J: A Modular Framework for Meta-heuristic Optimization - Source Code. *Consultado en* `https://github.com/felixreimann/opt4j`

[7] Código del Proyecto. Por Luis Serrano Hernández *Alojado en* `https://github.com/luiserh1/SP2D_Opt4j`