Portada

Contenido

1. Introducción

En este proyecto se quiere mejorar la eficiencia energética de edificios a través del uso de procedimientos metaheurísticos.

Un procedimiento heurístico se caracteriza por ser simple, a menudo basado en el sentido común o intuición, que se supone que ofrecerá una buena solución (aunque no necesariamente la óptima) a problemas difíciles, de un modo fácil y rápido.

Dicho esto, un metaheurístico tampoco garantiza la obtención de una solución óptima y también se basa en la aplicación de reglas relativamente sencillas, pero las técnicas metaheurísticas evitan aceptar los óptimos locales, si no que orientan la búsqueda en cada momento dependiendo de la evolución del proceso de búsqueda.

En este trabajo se creará un algoritmo metaheurístico que utiliza modelos de edificios para su funcionamiento. Estos modelos de edificios consisten en una serie de ficheros que se emplean en el programa de JEplus o EnergyPlus, que es un simulador de consumo energético de edificios, como se explicará más adelante (Ap. 4.3).

"EnergyPlus™ is a whole building energy simulation program that engineers, architects, and researchers use to model both energy consumption"

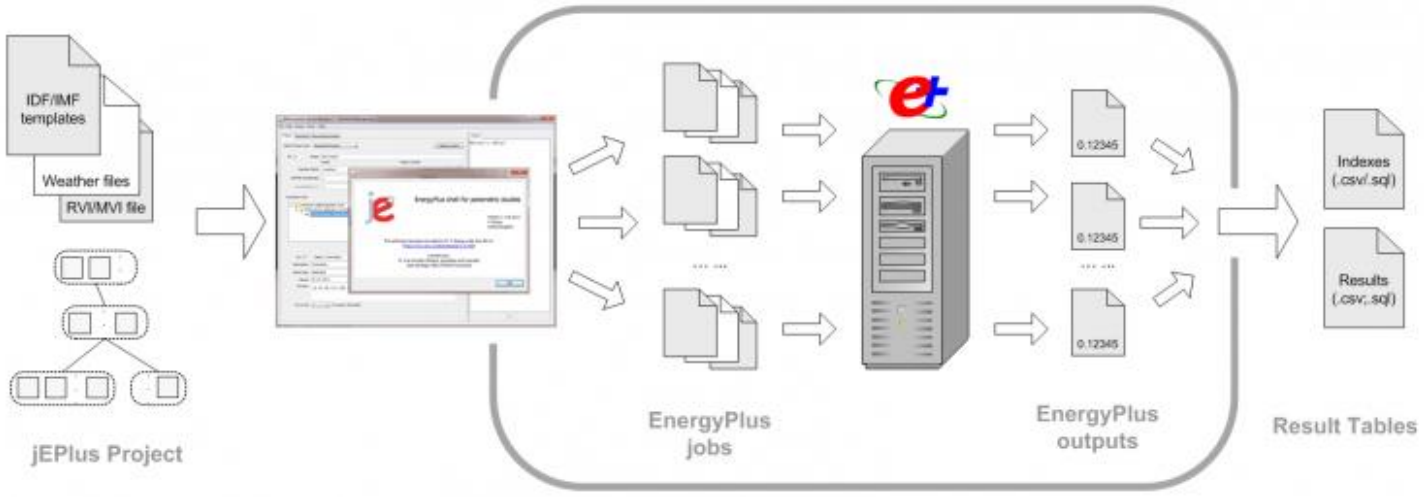


Ilustración 1. Esquema de funcionamiento de JEplus

A la hora de ejecutar nuestro programa se recogerá la información de esos archivos y se tendrán en cuenta estos tres factores:

-Parámetros no variables: Estos son pertenecientes a cada proyecto y son inmutables, como pueden ser la altitud de la zona donde se encuentra el edificio a construir, temperatura de la provincia, humedad del lugar a construir, promedio de lluvias anuales, horas de luz diarias u otra cantidad de posibles características climáticas.

-Parámetros variables: Estos son pertenecientes a cada edificio y pueden variar. Su modificación afecta a los resultados que vamos a obtener. Ejemplos de estos pueden ser: grosor de las paredes, orientación del edificio, materiales empleados, altura…

-Funciones a optimizar: Los parámetros anteriores se tienen en cuenta para calcular las funciones a optimizar. Estas son las salidas o factores energéticos relacionados con el edificio que se pretenden mejorar. Ejemplos de esto puede ser: Consumo eléctrico, gasto de calefacción, iluminación del interior, radiación solar o contaminación acústica entre otros muchos.

Para que se entienda fácil pongamos un ejemplo sencillo:

Se desea construir un edificio en Madrid. Los archivos del proyecto indican que existe un parámetro no variable, “En Madrid la temperatura media es de 21ºC”, un parámetro variable: “El ancho de la pared puede encontrarse entre 10cm y 80cm”, y unas funciones a optimizar: “Se quiere reducir la radiación solar” y “Se desea permitir que entre la temperatura del exterior”.

Entonces, en ese caso, puede que interese fijar el parámetro variable de grosor de pared en 10cm, porque con una pared poco aislante se cumple con que entre la temperatura del exterior, pero entonces podría entrar mucha radiación solar, pero al revés, si fijamos el ancho de pared a 80cm se puede bloquear mucha radiación, pero no se permitiría un correcto acceso de la temperatura del exterior.

Entonces, ¿Interesaría más una pared intermedia de 40cm? ¿Y sí se añaden otros parámetros variables que influyen entre sí? ¿Y si se añaden otras funciones que se quieren optimizar que se afectan mutuamente?

A través de nuestro algoritmo metaheurístico se resolveran todos estos problemas y se proporcionará la solución o soluciones que mejor cumpla con las salidas a optimizar, respetando siempre los parámetros recibidos.

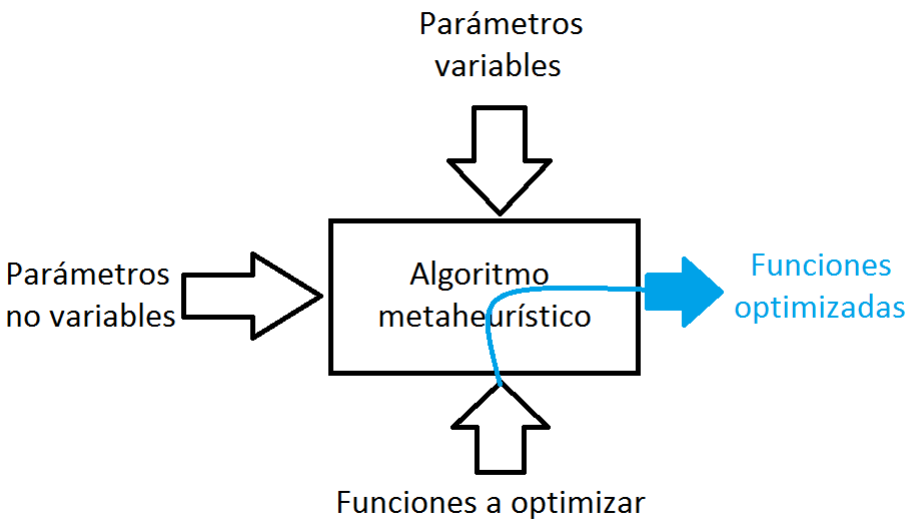


Ilustración 2. Esquema básico de las 3 entradas que recive nuestro programa

2. Objetivos

Respecto el trabajo a realizar, se busca conseguir una serie de objetivos tras su finalización.

El objetivo final consiste en óptimizar cada una de las funciones de salida relacionadas con el consumo energético del proyecto. Para conseguir esto se ha necesitado cumplir una serie de objetivos parciales:

- Entender el funcionamiento del programa JEplus: Ya que el algoritmo a crear va a tener que usarlo para obtener las funciones de salida correspondientes a los parámetros con los que se llame.

- Poder ejecutar JEplus no desde su interfaz si no desde el código: Se requiere crear un "job". Un "job" es un conjunto de datos que asignan un valor a los parámetros variables. Se reconocen por un identificador único. Se pueden agrupar en un conjunto de "jobs" que se envian a JEplus, que deberá devolver el valor de las funciones de salida especificadas.

- Generar un conjunto de "jobs" que exploren todas las opciones posibles dentro de los rangos de los parámetros variables: Esto se necesita para poder garantizar que si se exploran muchas y variadas combinaciones de los valores de los parámetros variables se podrá obtener un buen resultado de la optimización de las funciones de salida.

- Desarrollar y aplicar diferentes métodos de exploración y selección de valores de los rangos de los parámetros. Esto es necesario para aumentar las posibilidades de una exploración de los rangos completa y para obtener una diversidad de "jobs" más amplia. Además, cuando se generen los resultados finales se podra determinar cual es el método que mejor funciona y proporcione más cantidad y mejores soluciones.

- Crear un filtro para aplicar a las salidas obtenidas. Consistirá en, una vez se dispone del valor adecuado de los parámetros que forman una solución adecuada, se intenta realizar minúsculas modificaciones a los valores de los parámetros para comprobar si se producen mejoras. Esto sirve para intentar perfeccionar todavía más los resultados obtenidos.

3. Descripción algorítmica

En este apartado se explica el proceso de creación de nuestro algoritmo. Como se basa en la herramienta de JEplus, el primer paso consiste en conocer el funcionamiento del programa para poder usarlo posteriormente desde nuestro algoritmo. Una vez dominado esto, se hacen las configuraciones adecuadas para que pueda llamarse a JEplus desde nuestro programa.

A continuación, ya se puede proceder a crear nuestro algoritmo:

El primer paso que ejecuta nuestro programa es cargar los ficheros asociados al modelo del edificio para poder conocer sus caracteristicas y el rango de los valores de sus parámetros variables.

Estos ficheros son modificables por el usuario, aunque no son fáciles de entender ya que no presentan una estructura intuitiva y contienen grandes volumenes de datos, por tanto requieren un tiempo de familarización para poder comprender que información presenta cada uno y como manejarla para evitar posibles errores indeseados.

De esta manera se puede editar el número y los parámetros concretos asociados al proyecto.Se pueden añadir y eliminar tanto los fijos como los variables, y además filtrar las funciones de salida que no se deseen calcular.

Después, se generan un número específico de “jobs”, fijando un valor posible dentro del rango de cada parámetro variable para cada “job”.

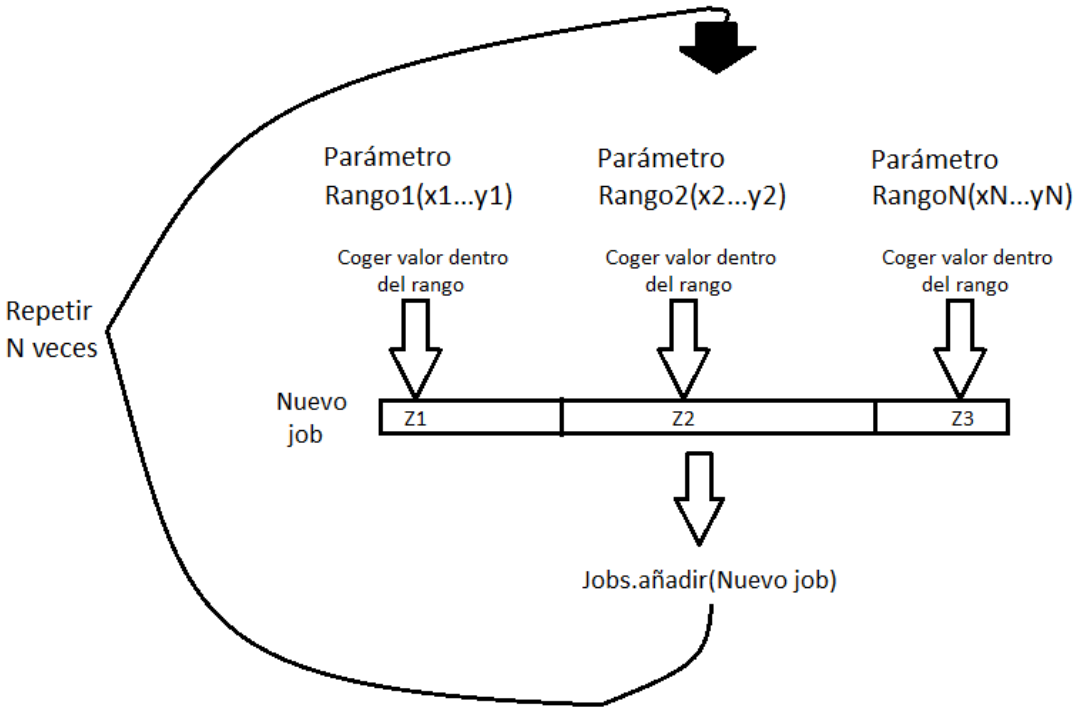


Ilustración 3. Proceso de generación de un "job"

Para elegir el valor a fijar dentro de cada rango se dispone de multitud de posibilidades, y a priori no tiene porque ser ninguna mejor que otra. Se opta por crear un módulo escalable al que se le puedan añadir infinitos métodos para calcular valores variados dentro de un rango.

En este trabajo se decide añadir dos métodos de extracción de un valor del rango. La explicación de la elección y funcionamiento de estos métodos se desarrolla en el apartado 3.1 perteneciente a los métodos constructivos.Tras finalizar el trabajo se podrá determinar cuál ha generado mejores resultados.

Una vez fijados todos los valores para cada parámetro de cada job , se agrupan los “jobs” y se envían a JEplus para que calcule el valor de las funciones energéticas asociadas a cada job. JEplus

devuelve estas salidas a nuestro programa y se comparan todas las soluciones recibidas entre sí, se filtran y se guardan solo las mejores.

Para realizar este filtrado, una salida se considera mejor que otra si y solo si mejora en todos sus factores energéticos. De la misma forma, se considera peor si consigue peores soluciones en todos sus factores a óptimizar. Si se produce el caso de que comparando dos soluciones, unos factores mejoran y otros empeoran no se puede detirminar cual es mejor y se deben almacenar ambas soluciones.

Esto es así debido a que no se cuenta con un "peso" asociado a cada factor energético, ya que no se puede determinar a priori que factor a optimizar tiene mayor importancia.

Por ejemplo, para una empresa o arquitecto puede ser más importante ahorrar 2 Kwh de electricidad que 2 Kwh de gas, pero para otros usuarios de esta herramienta puede suceder el caso contrario, ya sea porque el precio de cada suministro de energía varía en cada país o por cualquier otra razón. Por tanto no se puede asignar un orden o valor de importancia a cada posible función que se desee optimizar. Si no fuese así nuestro programa tan solo encontraría una solución, la mejor encontrada, pero de esta manera se ofrece al usuario todas las mejores posibiliades de solución encontradas.

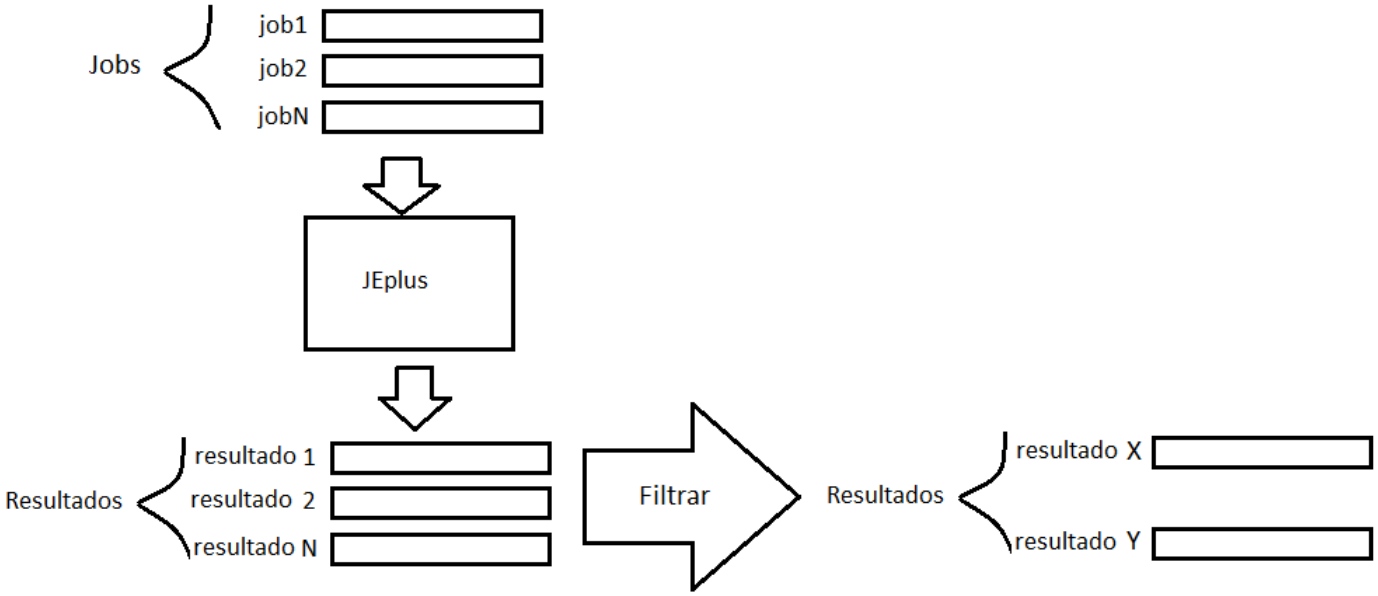


Ilustración 4. Proceso de obtención de las primeras soluciones

Una vez obtenidas y filtradas las mejores respuestas se podrían ofrecer al usuario y terminar el programa en este punto, pero para explorar más el rango de los parámetros y así detectar mejores posibles soluciones que no se hayan recogido, se procede a mejorar los resultados a partir de las soluciones temporales de las que ya se disponen. Para ello se realiza una búsqueda local. Básicamente consiste en realizar pequeñas modificaciones en los valores de los parámetros para ver como varía la solución, aunque esto se explica con profundidad en el apartado 3.2, "Búsqueda local".

Tras intentar mejorar las soluciones, si se consigue y se crean nuevas, hay que volver a compararlas con el resto de soluciones de las que no se podía determinar cual era mejor, y volver a descartar las peores. Si tras la fase de mejora se obtiene una solución que fuese mejor al resto, estas se descartan, ya sean nuevas o existentes.

3.1. Constructivos

En este trabajo se decide añadir dos métodos de extracción de un valor del rango. Tras finalizar el trabajo se podrá determinar cuál a generado mejores resultados:

- El primero es muy sencillo, dentro de ese rango se elige un número al azar. Como se crean un número elevado de “jobs” se puede decir que se garantiza cierta variabilidad a la hora de elegir valores. A este método se le llamara “Random” a partir de ahora.

- El siguiente método, se le denominará “Semi-Random”, es más sofisticado y además, evita los problemas de posible relativa poca variedad dentro de los valores pertenecientes a el rango. Para evitar esto y aumentar significativamente la variabilidad de valores, se empiza eligiendo un valor al azar, pero la diferencia está, en que, según donde se situe este valor dentro del rango, se asigna un peso mayor o menor para que el próximo número elegido se obtenga del segmento del rango donde aún no se hayan obtenido valores.

  Si el siguiente valor se vuelve a obtener de una zona próxima a los valores anteriores, el “peso” de las zonas lejanas aumenta y por tanto mayor probabilidad de que no se repitan valores cercanos. De esta manera se asegura mayor variabilidad y además, a mayor número de ”jobs” se generen, los pesos asignados se estabilizan y aumenta aún más la variabilidad de “jobs” creados.

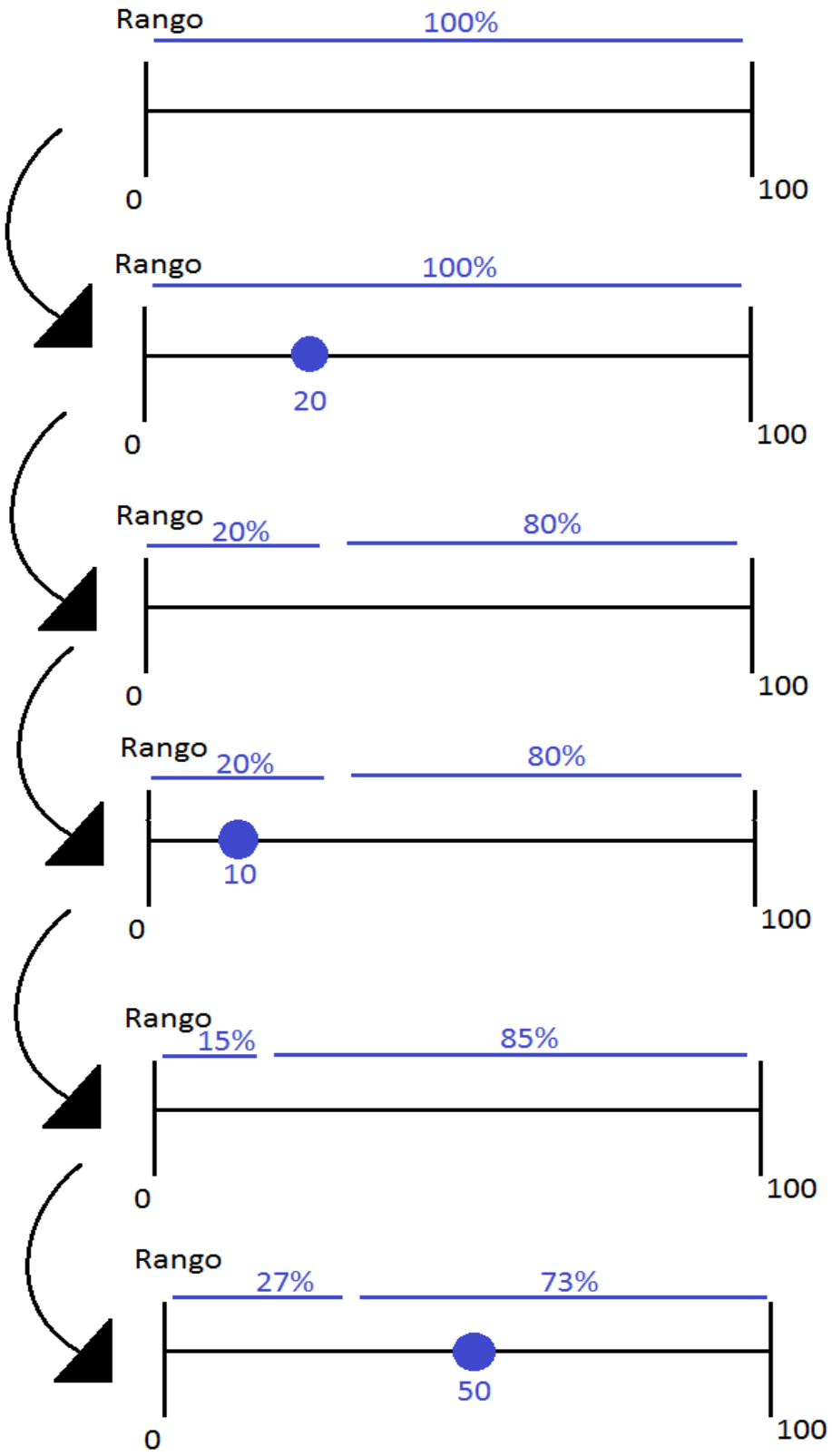


Ilustración 5. Ejemplo de funcionamiento de nuestro método "Semi-Random"

En esquema de la página anterior se muestra y ejemplifica como se generan los valores mediante el método “Semi-Random”.

En el primer paso las probabilidades de elegir un valor u otro son las mismas. Se elige un número al azar y obtenemos el 20. Ahora cambian las probabilidades.

$$ValorMaximoPosible - ((\sum valoresobtenidoshastaelmomento) \div Totalvaloresgenerados)$$

Aplicando la fórmula con el único valor hasta el momento obtenemos:

$$100 – (20 / 1) = 80.$$

Por tanto cuando se vaya seleccionar el siguiente valor, existe un 20% de posibilidades de que el valor provenga del primer segmento y un 80% del segundo segemento. Como en este ejemplo sencillo el rango se compone de los valores 0 hasta 100, coincide el porcentaje de probabilidades con los valores coincidentes comprendidos en esos porcentajes.

(Los valores comprendidos entre 0 y 20 tienen un 20% de probabilidades de ser elegidos. Los valores entre 20 y 100 tienen un 80% de probabilidades de ser elegidos. Por lo explicado, cuando el rango no es decimal esto no coincide)

En la siguiente iteracción del esquema, aunque solo exista un 20% de probabilidades de ser elegido, se escoge el valor 10. Por consiguiente se recalibran los pesos de los segmentos aplicando la fórmula:

$$100 – ((20+10) / 2) = 85.$$

A más iteracciones menor brusquedad en los cambios de los pesos. Y por tanto en este punto el peso del primer segmento se fija en 15. El segundo segmento aumenta y tiene 85% de probabilidades de que el valor elegido proceda de él.

Por último, el valor elegido es el 50.

$$100 – ((20+10+50) / 3) = 26,67.$$

Se comprueba como se redistribuyen los pesos, aunque se ha obtenido un valor alto, el cambio en las probabilidades es moderado, obteniendo un 27% y un 73% de probabilidades en el primer y segundo segmento respectivamente.

Este proceso del método “Semi-Random” se repite, para cada parámetro y tantas veces como número de “jobs” existan.

Una vez realizado este proceso para cada valor, de cada parámetro, de cada “job” creado, el siguiente paso consiste en enviar a JEplus todos los “jobs” que se han generado para que los ejecute y devuelva el valor de las funciones energéticas que se desean optimizar.

Una vez realizado esto, se comparan todas las salidas recibidas entre sí, se filtran y se guardan solo las mejores.

3.2. Búsqueda local

En este apartado se explica el proceso de mejora de las primeras soluciones obtenidas a través de una búsqueda local.

Búsqueda local es la base de muchos de los métodos usados en problemas de optimización.

Se puede ver como un proceso iterativo que empieza en una solución y la mejora realizando modificaciones locales.

Básicamente empieza con una solución inicial y busca en su vecindad por una mejor solución. Si la encuentra, reemplaza su solución actual por la nueva y continua con el proceso, hasta que no se pueda mejorar la solución actual

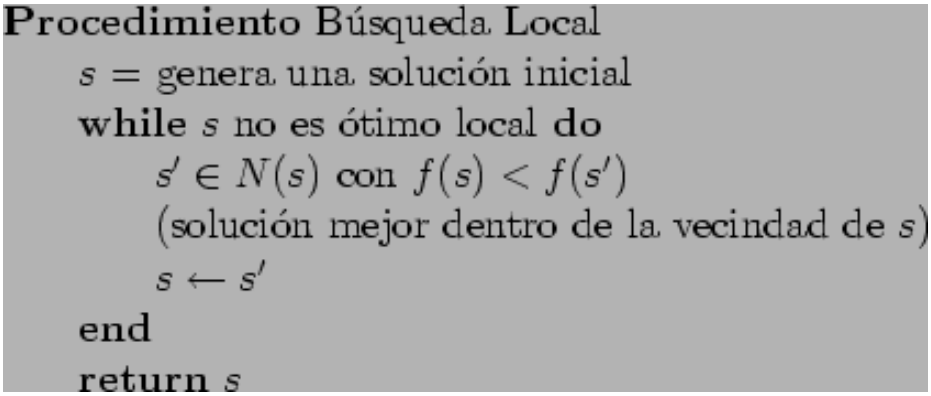


Ilustración 6. Pseudo código del proceso de búsqueda local

La vecindad son todas las posibilidades de soluciones que se consideran en cada punto.

El cómo se busca la vecindad y cuál vecino se usa en el reemplazo a veces se conoce como la regla de pivoteo (pivoting rule), que en general puede ser:

- *Seleccionar el mejor vecino de todos (best-improvement rule).*
- *Seleccionar el primer vecino que mejora la solución (first-improvement rule).*

Búsqueda local tiene la ventaja de encontrar soluciones muy rápidamente.

Su principal desventaja es que queda atrapada fácilmente en mínimos locales y su solución final depende fuertemente de la solución inicial.

https://ccc.inaoep.mx/~emorales/Cursos/Busqueda/node58.html

Para aplicar la búsqueda local a nuestras salidas, como cada solución del resultado se compone de, valor de los parámetros fijados y valores de los factores energéticos, se modifican ligeramente los valores de los parámetros eligiendo valores del rango cercanos para observar los cambios que producen en los factores energéticos.

Los cambios en los valores de los parámetros se desarrollan con especial atención. El proceso es el siguiente: El rango del parámetro se discretiza en “alpha” valores, para ello se framenta el rango del parámetro en “alpha” segmentos y se selecciona un valor de cada segmento.

El siguiente paso es comprobar el valor actual del parámetro y generar dos “jobs”, uno con el siguiente valor mayor del rango discretizado respecto del valor actual, y otro con el siguiente valor menor. Se ejecutan y se comparan los resultados para determinar cómo evoluciona la solución y hacía que dirección es mejor variar el parámetro.

Este proceso se realiza para todos los parámetros y de forma aleatoria en su orden para evitar no explorar todas las opciones posibles en el caso de que unos parámetros influyan sobre otros.

NOTA: EN RESULTADOS DECIR QUE ESTO ES IMPORTANTE PORQUE ES CIERTO QUE MEJORA MÁS.

El valor que debe tener el parámetro “alpha” dependerá del número de “jobs” a ejecutar. Si se generan muchos “jobs” es más probable que el rango de los parámetros se explore con mayor profundidad.

En ese caso el valor de “alpha” será mayor y por tanto se generan más particiones en el rango. Cuando se pruebe a mejorar las soluciones, la distancia que aumenta o disminuye el valor de un parámetro dentro de su rango es menor, y por tanto es menos probable saltarse valores útiles y nos aseguramos de una mayor diversidad en las pruebas de mejora de soluciones.

Por esto, es mejor idea que el valor del parámetro “alpha” sea dependiente del número de “jobs” a ejecutar, en vez de que sea un valor constante.

Con esta modificación en los parámetros se vuelve a enviar en forma de “job” a JEplus y cuando devuelve el resultado se comprueba si se han mejorado las funciones a óptimizar. Si tras varias iteracciones no se consigue una mejora, se opta por determinar que se tenía la mejor solución posible. Por el contario, si se consigue mejorar, se descarta la solución anterior y se guarda esta nueva.

3.3. GRASP

Nuestro algoritmo se basa en el esquema de un algoritmo GRASP. Aunque ya se ha explicado cómo funciona nuestro algoritmo, en este apartado explicaremos el funcionamiento general de un GRASP para entender mejor cuál es el porqué y la ventaja de aplicar este esquema en este trabajo.

GRASP (Greedy Randomized Adaptive Search Procedurees) una técnica de los años 80 desarrollada por Feo y Resende que tiene como objetivo resolver problemas difíciles en el campo de la optimización combinatoria. Esta técnica dirige la mayor parte de su esfuerzo a construir soluciones de alta calidad que son posteriormente procesadas para obtener otras aún mejores.

Los algoritmos GRASP son algoritmos de tipo iterativo en los que cada iteración incluye una fase de construcción de una solución y otra de postprocesamiento en la cual se optimiza la solución generada en la primera fase.

Se puede establecer una analogía con el problema de la Programación Lineal, donde primero se construye una solución factible y después se aplica el algoritmo Simplex. Sin embargo, en GRASP se le da bastante importancia a la calidad de la solución generada inicialmente.

La estructura básica de un algoritmo GRASP es la siguiente:

- *Mientras no se satisfaga el criterio de parada*
    1. *Construir una solución "greedy" aleatoria*
    2. *Aplicar una técnica de búsqueda local a la solución "greedy" aleatoria obtenida en el paso anterior (para mejorarla)*
    3. *Actualizar la mejor solución encontrada*

Se puede extender este algoritmo si se le añade un operador de mutación (mecanismo de generación de vecinos) al procedimiento GRASP básico:

- *Mientras no se satisfaga el criterio de parada*
    - *Solución = Solución "greedy" aleatoria*
    - *Repetir L veces*
        - *Solución = Búsqueda local (Solución)*
        - *Actualizar la mejor solución (si corresponde)*
        - *Solución = Mutación(Solución)*

http://elvex.ugr.es/software/nc/help/spanish/nc/clustering/GRASP.html

Fernando Berzal Galiano Computer Engineer PhD in Computer Science ACM Senior Member & IEEE Computer Society Member.

http://www.grafo.etsii.urjc.es/sites/default/files/papers/CAEPIA%20%282%29_3.pdf

Como su nombre indica, se trata de una adaptación del algoritmo Greedy en dónde cada una de las soluciones se construye aleatoriamente (dentro de unos límites establecidos). Tanto el algoritmo Greedy como el GRASP consisten en una búsqueda de soluciones de manera constructiva, en lugar de avariciosamente como es el caso de Greedy en donde las tareas con mejores atributos van siendo seleccionadas en la secuencia.

Debido a que en numerosas ocasiones la toma de decisiones debe ser realizada en tiempos razonables, es necesario desarrollar una heurística que intente disminuir al máximo los tiempos de computación sin penalizar mucho la solución del problema.

http://bibing.us.es/proyectos/abreproy/70317/fichero/Capitulo+6.pdf

Servidor de la Biblioteca de Ingeniería. Universidad de Sevilla

```
{
Procedure GRASP (Max-Iteration)
Read Input();
For k = 1 to Max-Iteration (e.g., 500)
    Solution ← Greedy Randomised Construction.
    If Solution is not feasible then
        Solution ← Repair(Solution).
    End if
    Solution ← Local Search(Solution).
    Update Solution(Solution,Best solution).
    Update Pareto Archive.
End for
Return Pareto Archive.
End GRASP
}
```

Ilustración 7. Pseudo código del algoritmo genérico GRASP

http://www.scielo.org.za/scielo.php?script=sci_arttext&pid=S2224-78902012000300008

The Scientific Electronic Library Online (SciELO) SA is South Africa's premier open-access (free to access and free to publish) searchable full-text journal database in service of the South African research community

4. Descripción informática

4.1. Lenguaje

El lenguaje de programación empleado es Java. Esto se debe esencialmente a que es un lenguaje con el que tenemos mucha experiencia y que el programa de JEplus al que se llama desde nuestro algoritmo también está implementado en java y se utiliza su archivo ".jar" como libreria, así nos aseguramos evitar problemas de posibles incompatibilidades y facilitar su uso.

4.2. Entorno

El entorno utilizado es NetBeans, al igual que el lenguaje de programación tenemos mucha experiencia sobre él y es el que utilizo para los proyectos relacionados con el grado. Por esa sencilla razón es el utilizado.

4.3. Software utilizado

Para realizar el trabajo y poder ejecutar nuestro algoritmo se ha requerido del siguiente software adicional:

- EnergyPlus: "EnergyPlus™ is a whole building energy simulation program that engineers, architects, and researchers use to model both energy consumption". Cabe destacar sus ficheros asociados, con extensión ".idf" o ".imf" y los ficheros ".rvi" o ".rvx". Son los utilizados por EnergyPlus y JEplus para ejecutar los "jobs". Contienen los parámetros del modelo del edificio y las funciones energéticas a optimizar. Son modificables para eliminar parámetros no deseados del calculo de las funciones de salida, que también se pueden filtrar.

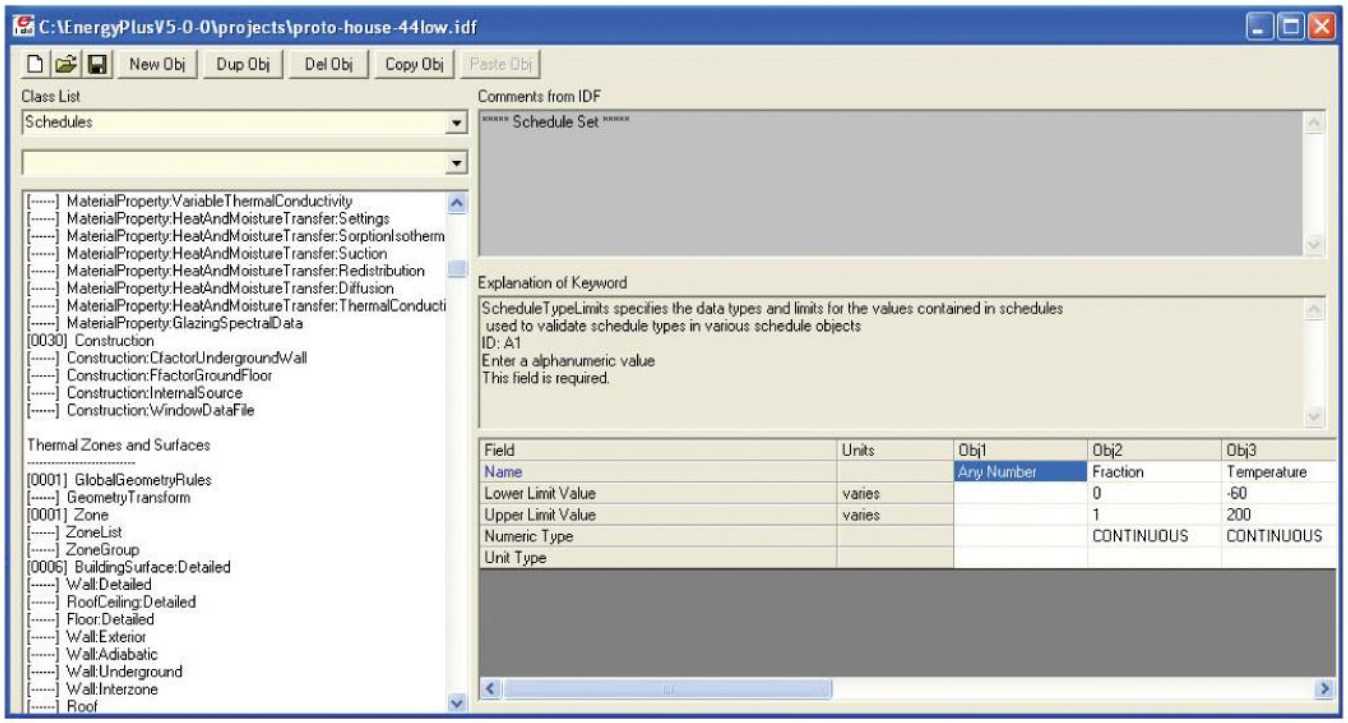


Ilustración 8. Interfaz de EnergyPlus con el archivo de extensión ".idf" del modelo del edificio abierto

- JEPlus: Es una interfaz que utiliza EnergyPlus para añadirle parametrización y poder ver todos los resultados. Su funcionamiento consiste en especificar los ficheros con la información del proyecto y los parámetros variables que vamos a utilizar, así como todos los posibles valores que pueden adquirir estos parámetros. La especificación de todo esto es lo que llaman en JEplus "jobs". El resultado de esto es una enorme cantidad de archivos de extensión ".csv" con todas las salidas generadas.

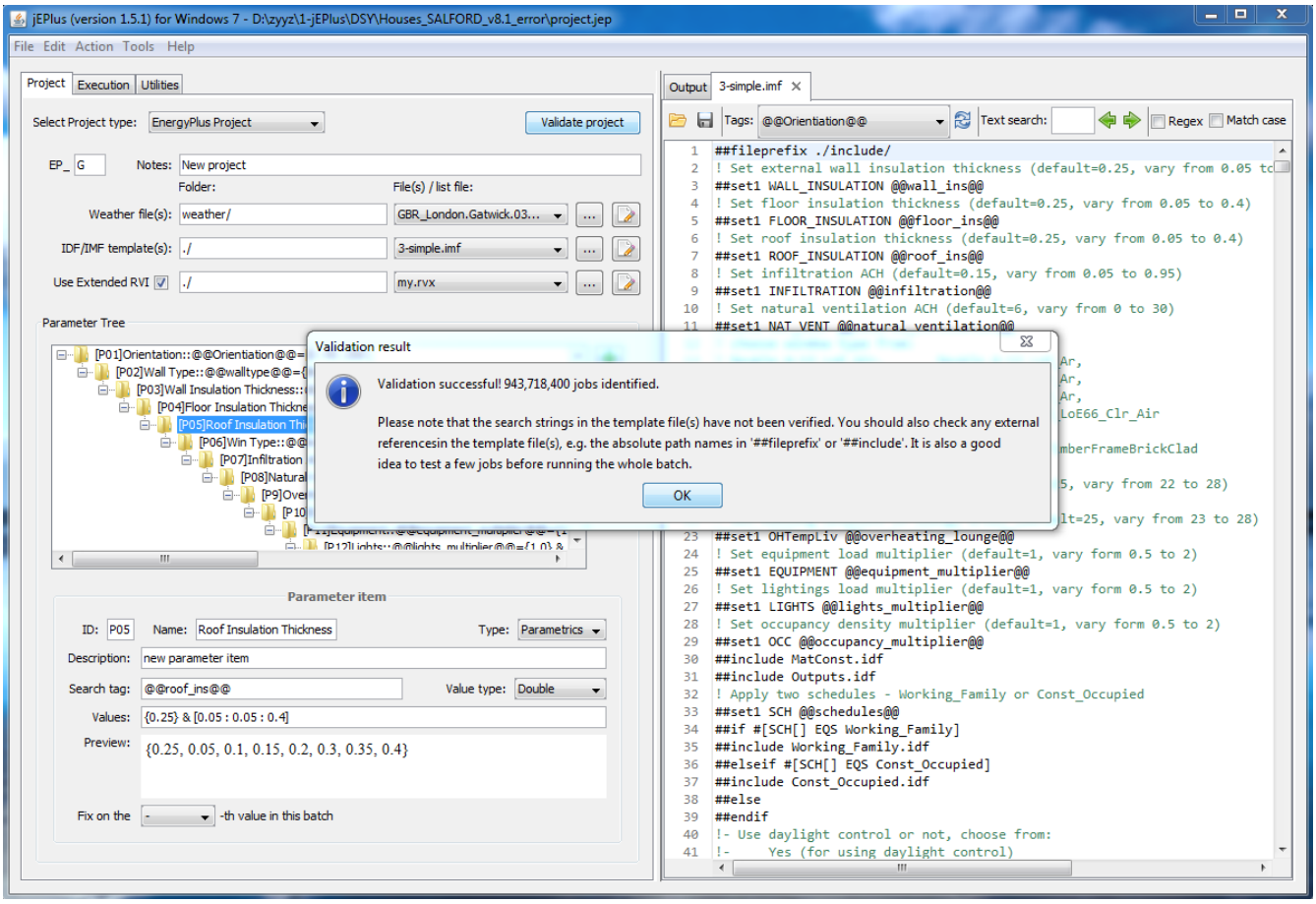


Ilustración 9. JEplus siendo usado desde su interfaz gráfica

- Adiccionalmente se ha utilizado la herramienta de “Github” para realizar el control de versiones. En el caso de que el proyecto situado en local se modificase de tal forma que no se encuenta forma de recuperar su estado anterior estable o simplemente se perdiese la información almacenada, se ha utilizado “Github” como software adiccional para realizar copias de seguridad y así mantener siempre una versión estable del proyecto.

- Para construir los modelos UML del siguiente apartado se ha utilizado la herramienta de “Modelio”. Es una herramienta gratuita, fácil de utilizar y que crea esquemas muy limpios.

- Para visualizar los resultados de la ejecución de nuestro programa se ha utilizado “Plotly”: *“Plotly creates leading open source tools for composing, editing, and sharing interactive data visualization via the Web”.* https://plot.ly/

4.4. UML

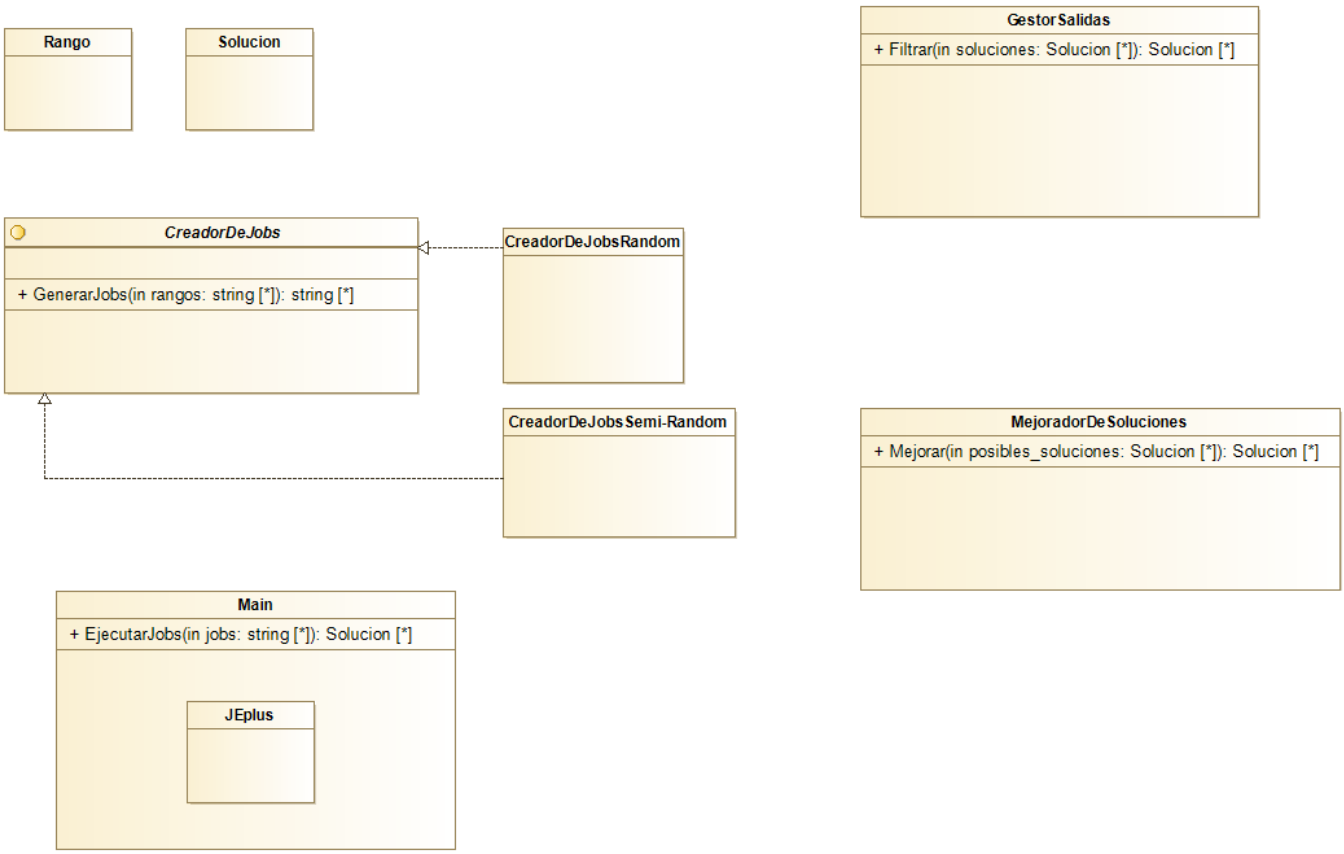


Ilustración 10. Diagrama UML de las clases creadas

En este apartado se explica las clases creadas y utilizadas en nuestro programa, así como la parte del algoritmo que ejecuta cada una y cómo funcionan.

- "Rango": Es una clase creada para facilitar la gestión del rango de los parámetros del modelo del edificio. Cómo cada parámetro posee un valor mínimo y un máximo, por cada parámetro especificado, se lee en los ficheros y se crea una variable de tipo Rango que guarda en dos atributos de tipo "double" su valor mínimo y máximo.

- "Solucion": Es la clase que se utiliza una vez se reciban las primeras salidas de JEplus. Se compone de dos atributos:

    - "parametros": es un array de strings que contiene al "job". Especifica su identificador único y el valor fijado de los parámetros variables.

    - "solucion": Es un array de strings de tanta longitud como funciones de salida a optimizar haya. Es el resultado que adquieren las funciones a optimizar cuando se envian a JEplus los parámetros con el valor guardado en el atributo "parametros"

Además de para mostrar al arquitecto o usuario del programa que decisiones debe tomar a la hora de construir el edificio, es importante guardar la información de “parámetros” ya que es necesario saber que valores tenían los parámetros que formaron la solución por si esta se quiere intentar mejorar.

- “CreadorDeJobs”: Esta interfaz, como se comenta en el apartado 3.1, corresponde al módulo de generación de “jobs”. Como existen infinitos métodos para crear “jobs” y así dar un valor a los parámetros variables, y se desea generar “jobs” muy diversos para explorar todo el abanico de posibles soluciones, se crea, por tanto, esta interfaz para que sea implementada por tantas clases como se deseen. Cada clase corresponde a un método de generación de “jobs” distinto.

  En este trabajo se decide crear dos clases que implementen el método de “GenerarJobs”. Este método recibe el número de jobs a crear y un string de “Rangos” con el rango de cada parámetro. Su salida consiste en un array de de “jobs”. El “job” se representa a su vez como un array de strings con su identificador único y el valor fijado de los parámetros variables calculado en “GenerarJobs”.

  Las dos clases son “CreadorDeJobsRandom” y “CreadorDeJobsSemi-Random” que se corresponden con los métodos “Random” y “Semi-Random” respectivamente, explicados en el apartado 3.1.

- “GestorSalidas”: Esta clase se encarga de recibir las soluciones creadas. recibe un array de “Solucion” con todas las soluciones generadas por JEplus (Una por cada job)

  Esta clase crea un nuevo array que es el que devolverá en su método que realiza el filtrado de soluciones. En este método, mediante un bucle que explora cada “Solucion” del array que recibe como argumento, realiza una serie de comprobaciones y decide si el método se filtra o no. Dentro de cada “Solucion” recorre a su vez los valores de las funciones energéticas obtenidos. Si todos los valores son peores a los contenidos en la anterior “Solucion” del array de salida, esta “Solucion” se descarta. Si por el contrario mejora en todos los valores de las funciones de optimización, se compara con las soluciones ya filtradas,se descartan las peores y se guarda esta nueva.

  Una vez recorridas todas las soluciones iniciales, se devuelve un array que contiene las soluciones optimales.

- “MejorarSoluciones”: Esta clase recibe las soluciones ya filtradas de la clase “GestorSalidas”. Como su nombre indica, se intentarán mejorar. Para ello, se crea un bucle que examina todas las soluciones. Para cada “Solucion” se recorre a su vez el array del atributo “parametros”. Este array se explora en orden aleatorio porque se va a proceder a cambiar el valor de cada parámetro y puede ocurrir que el orden afecte cuando se calcule el valor de las fuciones a optimizar. Gracias a esto se evita restringir la posibilidad de encontrar soluciones válidas.

A continuación, como se conoce el rango de cada parámetro, se divide en “alpha” partes como se explica en el apartado 3.2, y se crean dos arrays de strings con el mismo valor que el atributo “parametros” que se este recorriendo. Estos dos “jobs” cambiarán el valor del parámetro que se encuentre recorriendo el bucle, uno con el siguiente valor mayor del rango discretizado respecto del valor actual, y otro con el siguiente valor menor. Se envian a JEplus, se ejecutan y se comparan las salidas para determinar cómo evoluciona la solución y hacía que dirección es mejor variar el parámetro.

Suponiendo que es mejor la “Solucion” en la que se aumenta el parámetro, se itera en un nuevo bucle hasta tres veces cambiando el valor del parámetro actual por el siguiente mayor de su rango discretizado. Se ejecuta el job contenido en el atributo “parametros” de esta “Solucion” y se compara el atributo “solucion”. Si mejora se guarda esta “Solucion” con los parámetros modificados, y si no mejora tras tres iteraciones se deja como estaba. (Se ha optado por parar de mejorar si no lo consigue en tres iteraciones por considerarlo razonable. Igualmente válido habría sido 5 iteraciones pero cabe considerar que los tiempos de ejecución aumentan y tampoco se garantiza que tras 5 iteraciones sí mejore)

En ambos casos se siguen recorriendo todos los parámetros y repitiendo la misma operación por si mejorase nuevamente al modificar más parámetros.

Esto se realiza para todas las soluciones y se devuelven en el mismo array de entrada pero habiendose modificado (solo si han mejorado) las variables de tipo “Solucion” que conforman el array.

Este es sin duda la clase más costosa computacionalmente y la que tarda más en ejecutar ya que su método “principal” tiene una complejidad O(n3)

- “Main”: TERMINAR ESTE Y CONTINUAR POR RESULTADOS.

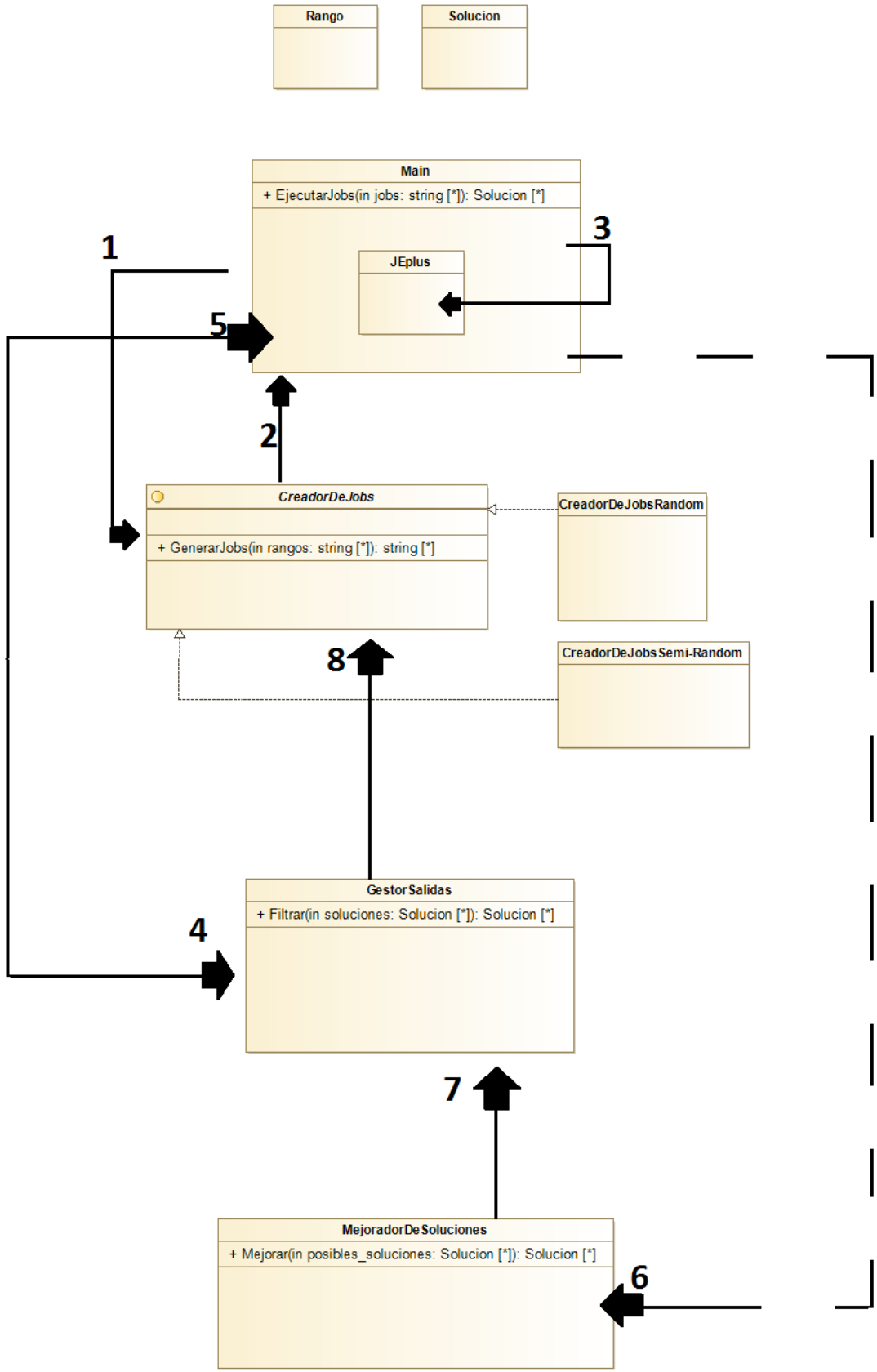


Ilustración 11. Orden de llamadas entre las clases creadas

5. Resultados

el valor del “alpha ”es muy importante y en un primer lugar se desconcía en base a que factor asignarlo. Se pensaba como una constante..primero 100..luego 500..primer caso muchas soluciones se repetian…segundo apenas había cambios…solución asignarlo en base a nº de jobs……

. Una asignación de bienes es óptimo en el sentido de Pareto (o Pareto eficiente) cuando no hay posibilidad de redistribución de una manera en la que al menos una persona estaría mejor, mientras que ningún otro individuo terminase peor.

Vilfredo Pareto en su libro “Manuale di economia politica”

Soluciones totales: 280

Sol random:103

Sol random Mejoradas: 45

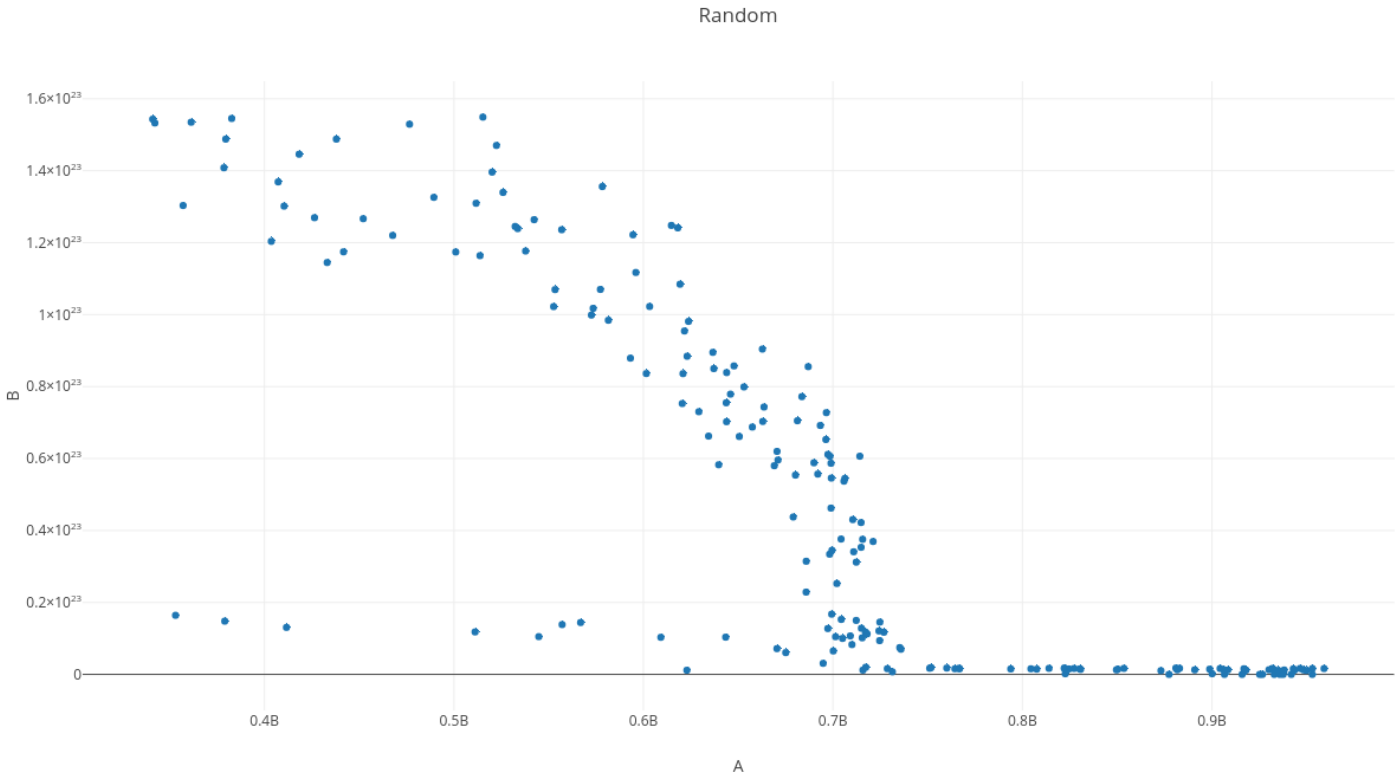
Sol semi-random: 88

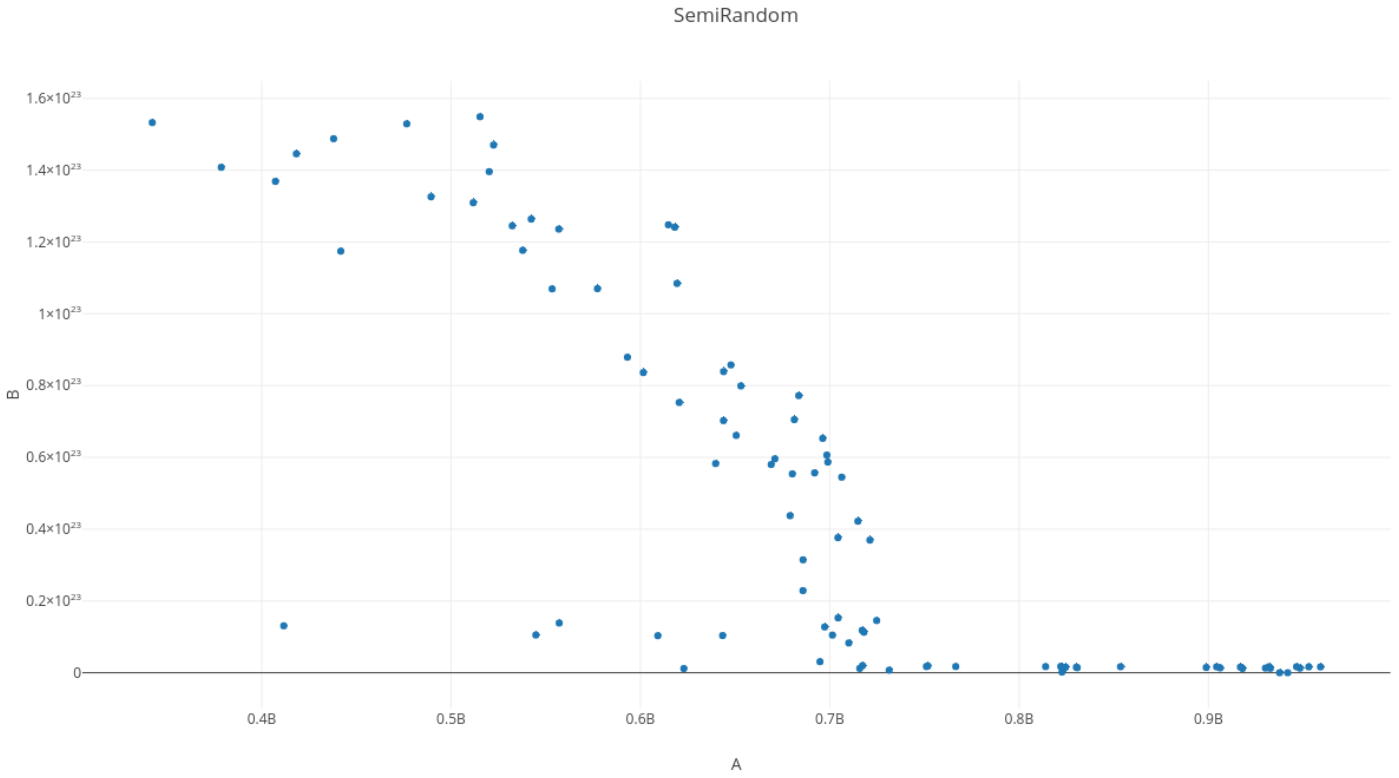
Sol semi-random Mejoradas: 44

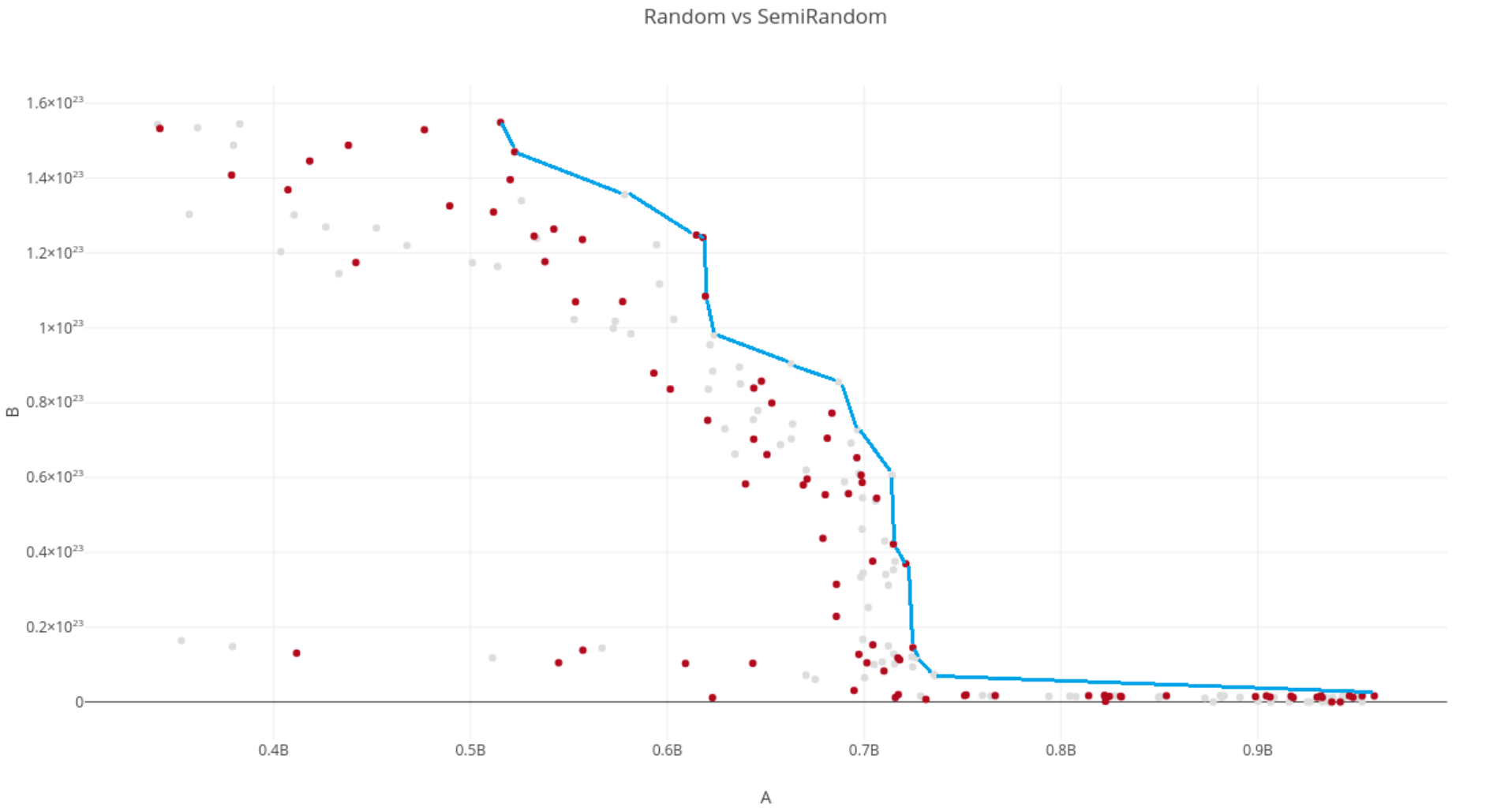
5.1. Experimentos

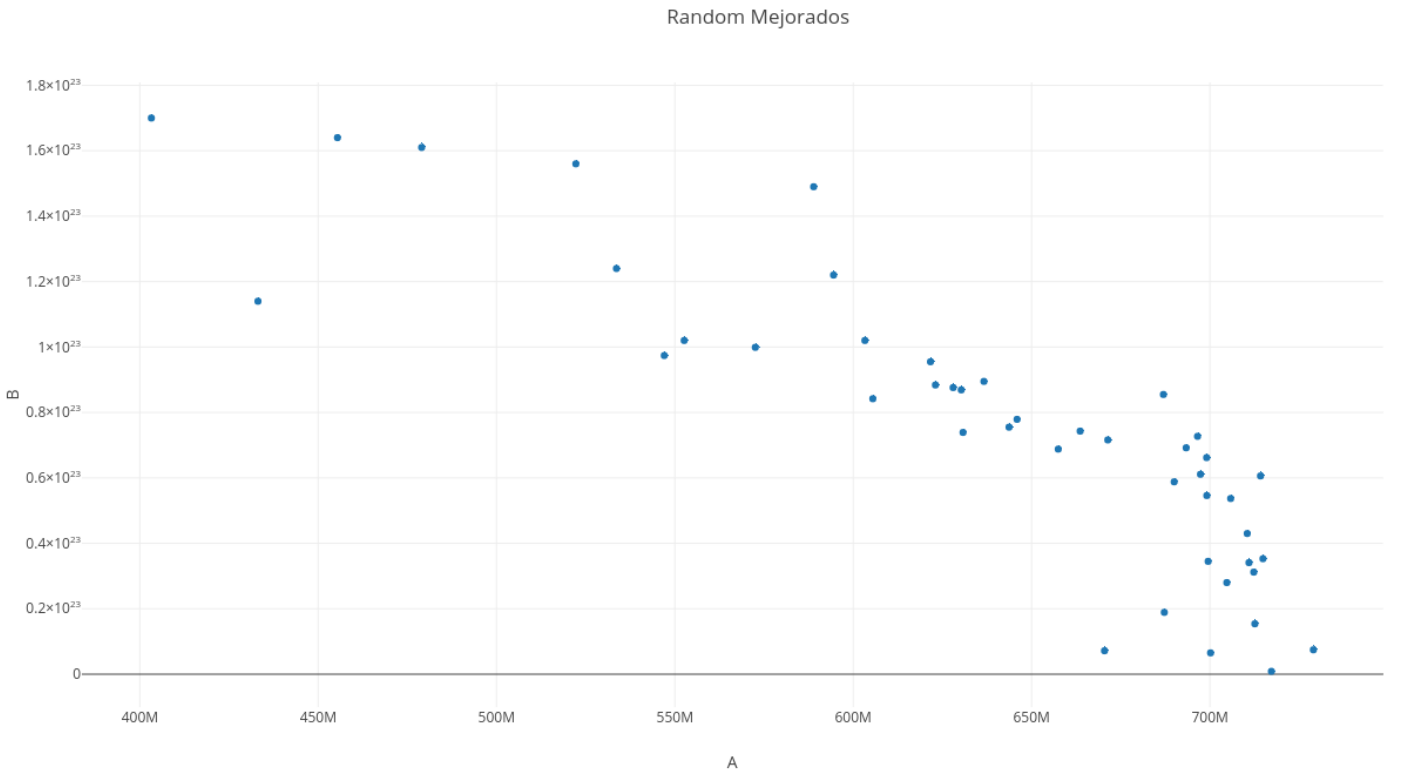
5.1.1. Preliminares

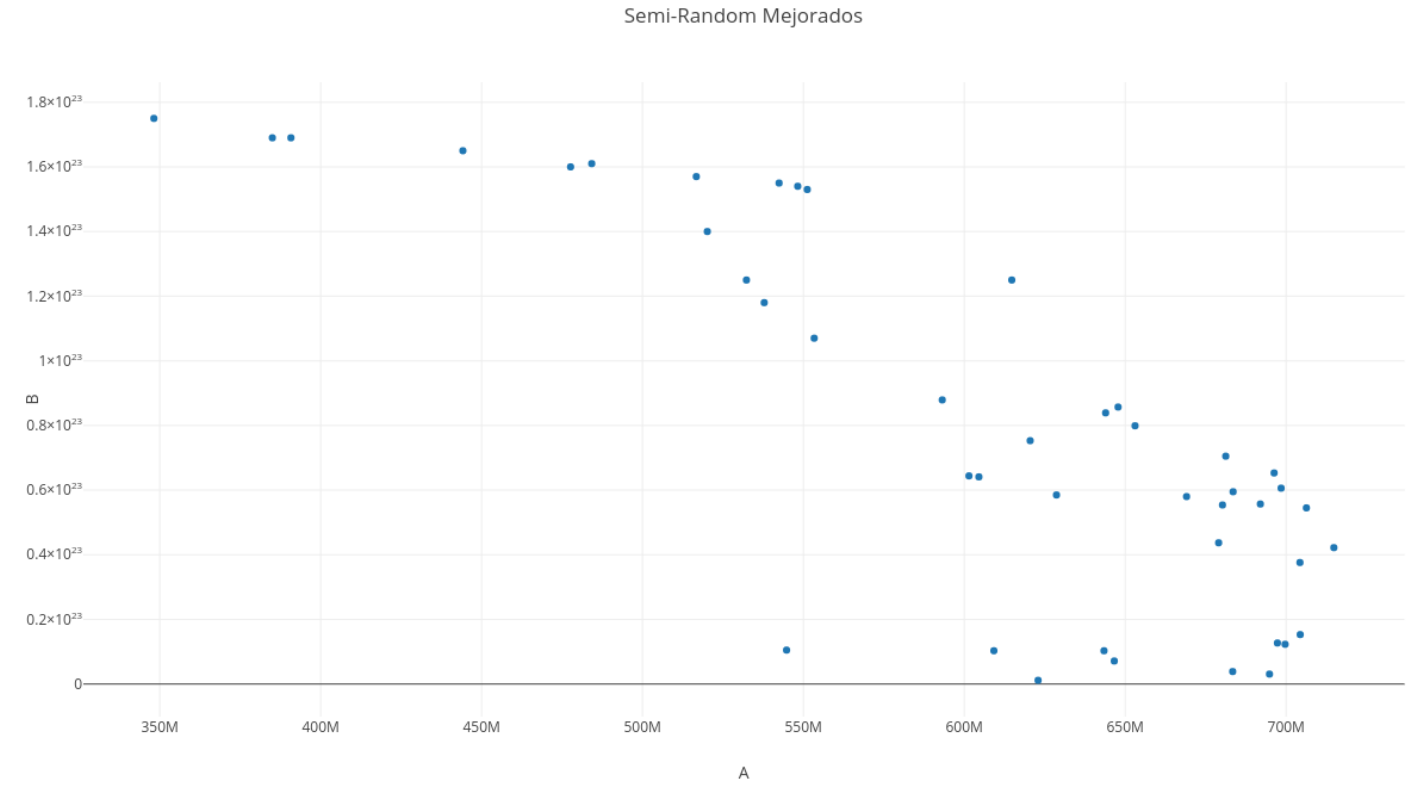
RANDOM

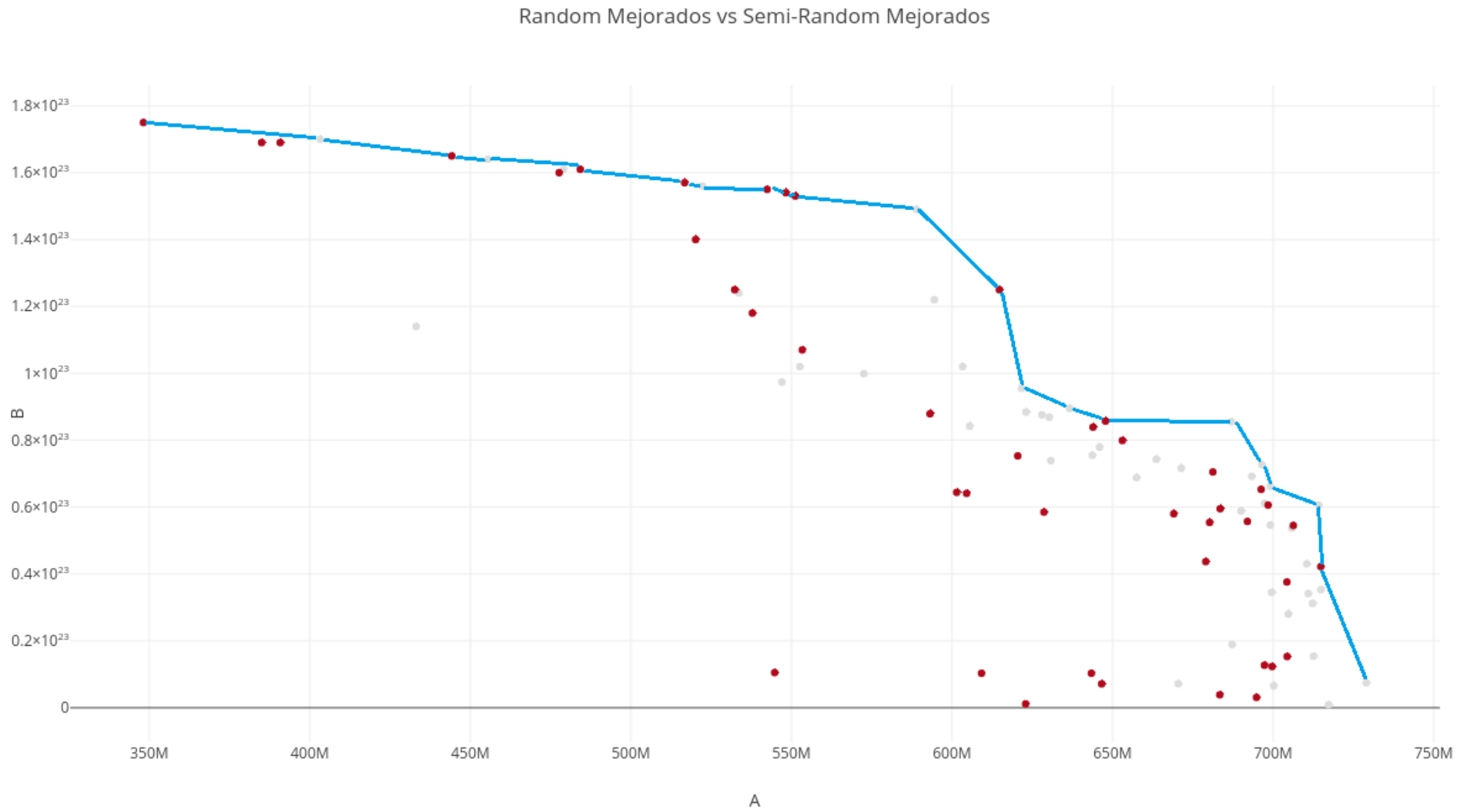

Random
1.6×10²³
1.4×10²³
1.2×10²³
1×10²³
0.8×10²³
0.6×10²³
0.4×10²³
0.2×10²³
0
B
0.4B
0.5B
0.6B
0.7B
0.8B
0.9B
A


SemiRandom
1.6×10²³
1.4×10²³
1.2×10²³
1×10²³
0.8×10²³
0.6×10²³
0.4×10²³
0.2×10²³
0
B
0.4B
0.5B
0.6B
0.7B
0.8B
0.9B
A


Random vs SemiRandom
1.6×10²³
1.4×10²³
1.2×10²³
1×10²³
0.8×10²³
0.6×10²³
0.4×10²³
0.2×10²³
0
B
0.4B
0.5B
0.6B
0.7B
0.8B
0.9B
A


Random Mejorados
1.8×10²³
1.6×10²³
1.4×10²³
1.2×10²³
1×10²³
0.8×10²³
0.6×10²³
0.4×10²³
0.2×10²³
0
B
400M
450M
500M
550M
600M
650M
700M
A


Semi-Random Mejorados
1.8×10²³
1.6×10²³
1.4×10²³
1.2×10²³
1×10²³
0.8×10²³
0.6×10²³
0.4×10²³
0.2×10²³
0
B
350M
400M
450M
500M
550M
600M
650M
700M
A


Random Mejorados vs Semi-Random Mejorados
1.8×10²³
1.6×10²³
1.4×10²³
1.2×10²³
1×10²³
0.8×10²³
0.6×10²³
0.4×10²³
0.2×10²³
0
B
350M
400M
450M
500M
550M
600M
650M
700M
750M
A

5.1.2. Finales

6. Conclusiones y trayectorias futuras

20h 58min 45 seg

"CreadorDeJobs" se deja la interfaz para que sea implementada por todas las clases que se quieran.

+ todo lo de la hoja

7. Bibliografía

https://github.com/Yunai-Bajo-Gallego/TFG

https://energyplus.net/ : Definicion energyplus

http://www.jeplus.org/wiki/doku.php : Imágenes

https://www.buildingenergysoftwaretools.com/software/jeplus : Imágenes

https://www.homepower.com/ : Imágenes

https://plot.ly/#/ Para las graficas

https://ccc.inaoep.mx/~emorales/Cursos/Busqueda/node58.html : instituto nacional de astrofísica óptica y electrónica. Para definición de búsqueda local.

http://elvex.ugr.es/software/nc/help/spanish/nc/clustering/GRASP.html :GRASP

http://www.grafo.etsii.urjc.es/sites/default/files/papers/CAEPIA%20%282%29_3.pdf :GRASP

http://bibing.us.es/proyectos/abreproy/70317/fichero/Capitulo+6.pdf :GRASP

http://www.scielo.org.za/scielo.php?script=sci_arttext&pid=S2224-78902012000300008 :GRASP