

**ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA**

**GRADO EN INGENIERÍA INFORMÁTICA**

**Curso Académico 2017/2018**

**Trabajo Fin de Grado**

**MEJORA DE LA EFICIENCIA ENERGÉTICA DE LOS EDIFICIOS A TRAVÉS DEL USO DE METAHEURÍSTICAS.**

**Autor**: Yunai Bajo Gallego

**Director**: Jesús Sanchez-Oro Calvo

Contenido

[1. Introducción 2](#_Toc517024315)

[2. Objetivos 4](#_Toc517024316)

[3. Descripción algorítmica 5](#_Toc517024317)

[3.1. Constructivos 8](#_Toc517024318)

[3.2. Búsqueda local 12](#_Toc517024319)

[3.3. GRASP 14](#_Toc517024320)

[4. Descripción informática 16](#_Toc517024321)

[4.1. Lenguaje 16](#_Toc517024322)

[4.2. Entorno 16](#_Toc517024323)

[4.3. Software utilizado 17](#_Toc517024324)

[4.4. UML 20](#_Toc517024325)

[5. Resultados 27](#_Toc517024326)

[5.1. Experimentos 29](#_Toc517024327)

[5.1.1. Preliminares 30](#_Toc517024328)

[5.1.2. Finales 36](#_Toc517024329)

[6. Conclusiones y trayectorias futuras 39](#_Toc517024330)

[7. Bibliografía 40](#_Toc517024331)

# 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”* (1)

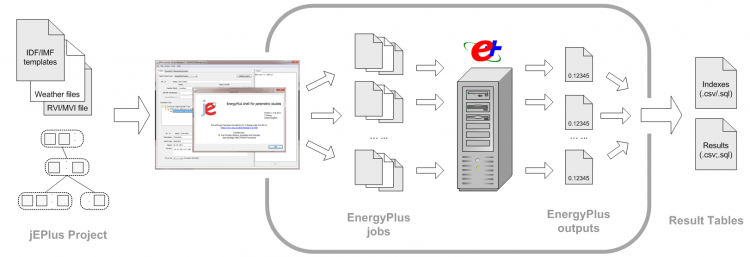


Ilustración 1. Esquema de funcionamiento de JEplus. (2)

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 si 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 resolverán todos estos problemas y se proporcionará la solución o soluciones que mejor cumpla con las salidas a optimizar, respetando siempre los parámetros recibidos.

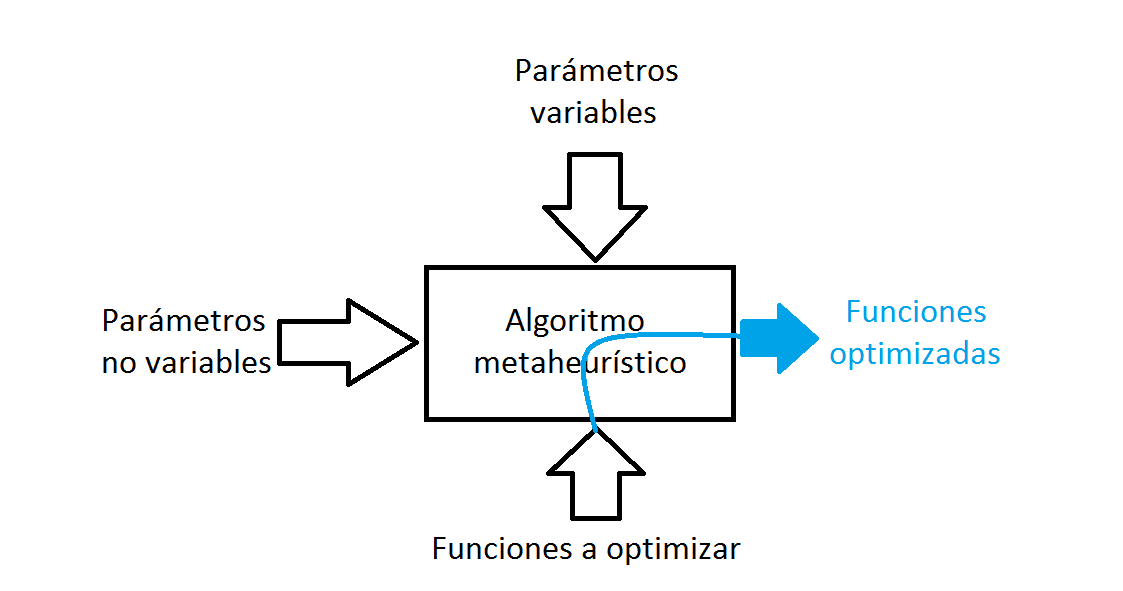


Ilustración 2. Esquema básico de las 3 entradas que recibe nuestro programa.

# Objetivos

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

El objetivo final consiste en optimizar 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 envían 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 podrá determinar cuál 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.

# 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 características 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 volúmenes de datos, por tanto, requieren un tiempo de familiarización para poder comprender que información presenta cada uno y cómo 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”.

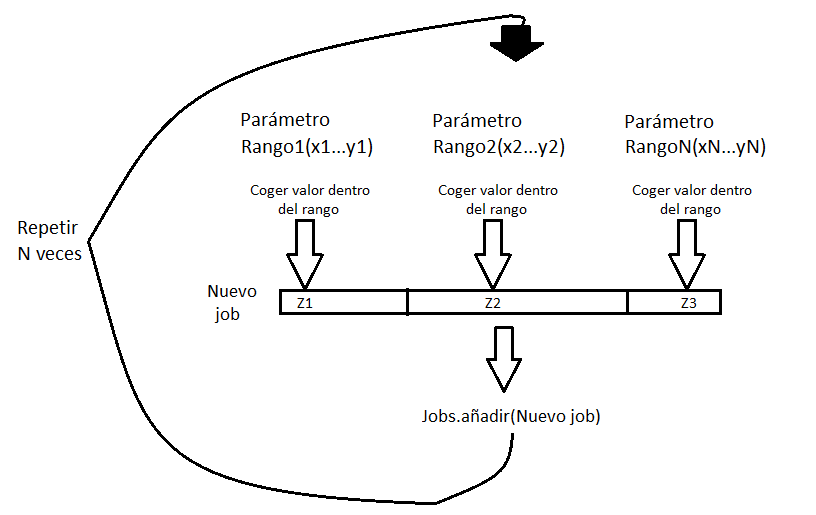


Ilustración 3. Proceso de generación de un "job".

Para elegir el valor a fijar dentro de cada rango se dispone de multitud de posibilidades, y a priori no tiene por qué 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 optimizar. Si se produce el caso de que, comparando dos soluciones, unos factores mejoran y otros empeoran no se puede determinar 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 posibilidades de solución encontradas.

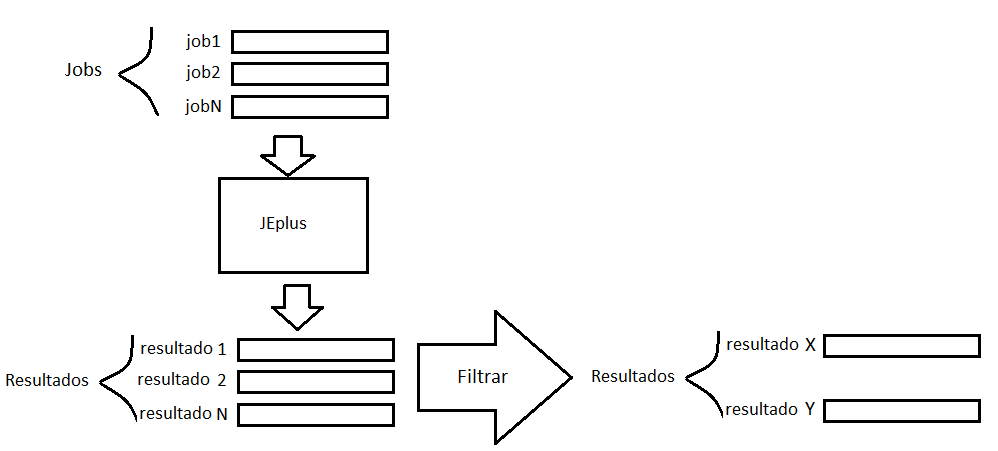


Ilustración 4. Proceso de obtención de las primeras soluciones.

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 cuál 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.

## Constructivos

En este trabajo se decide añadir dos métodos de extracción de un valor del rango. Tras finalizar el trabajo se podrá determinar cuál ha 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 empieza eligiendo un valor al azar, pero la diferencia está, en que, según donde se sitúe 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.

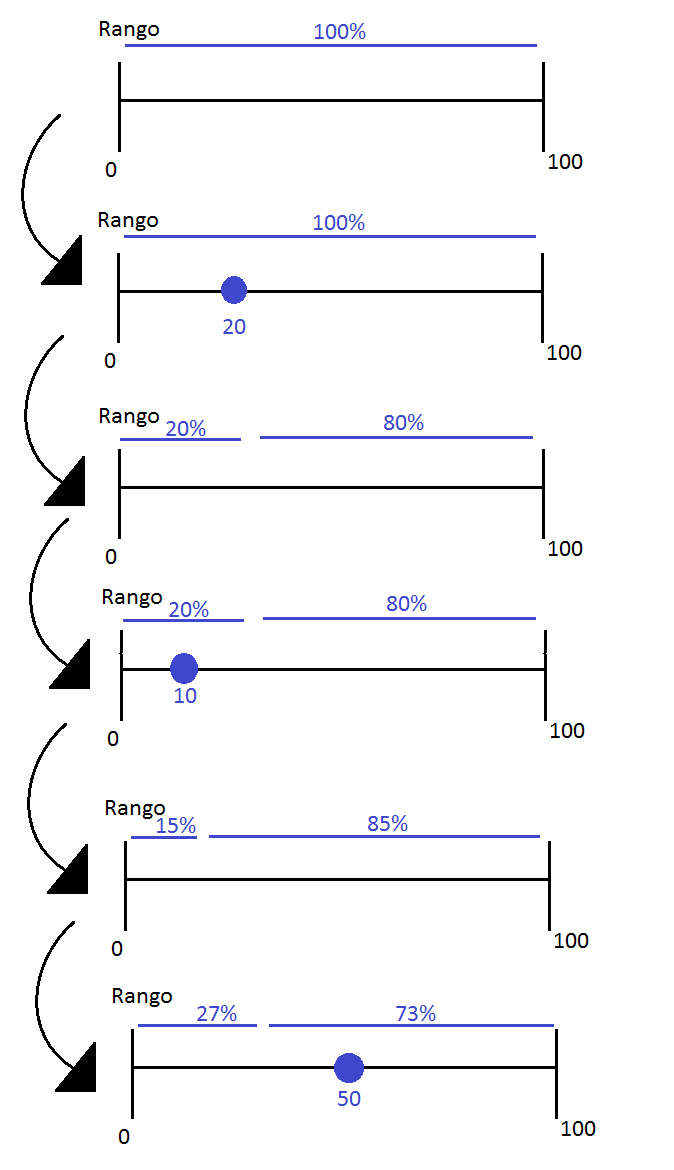


Ilustración 5. Ejemplo de funcionamiento de nuestro método "Semi-Random".

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.



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 segmento. 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 iteración 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 iteraciones 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 cómo 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.

## 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.”*

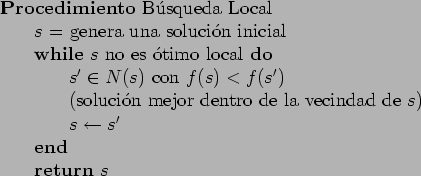


Ilustración 6. Pseudo código del proceso de búsqueda local.

*“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.”* (3)

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 fragmenta 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.

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 optimizar. Si tras varias iteraciones no se consigue una mejora, se opta por determinar que se tenía la mejor solución posible. Por el contrario, si se consigue mejorar, se descarta la solución anterior y se guarda esta nueva.

## 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 por qué 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)* (4)(5)

*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.”* (6)

**

Ilustración 7. Pseudo código del algoritmo genérico GRASP. (7)

# Descripción informática

## 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 librería, así nos aseguramos evitar problemas de posibles incompatibilidades y facilitar su uso.

## 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.

## 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.*

*EnergyPlus is a console-based program that reads input and writes output to text files. It ships with a number of utilities including IDF-Editor for creating input files using a simple spreadsheet-like interface, EP-Launch for managing input and output files and performing batch simulations, and EP-Compare for graphically comparing the results of two or more simulations.”* (1)

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 cálculo de las funciones de salida, que también se pueden filtrar.

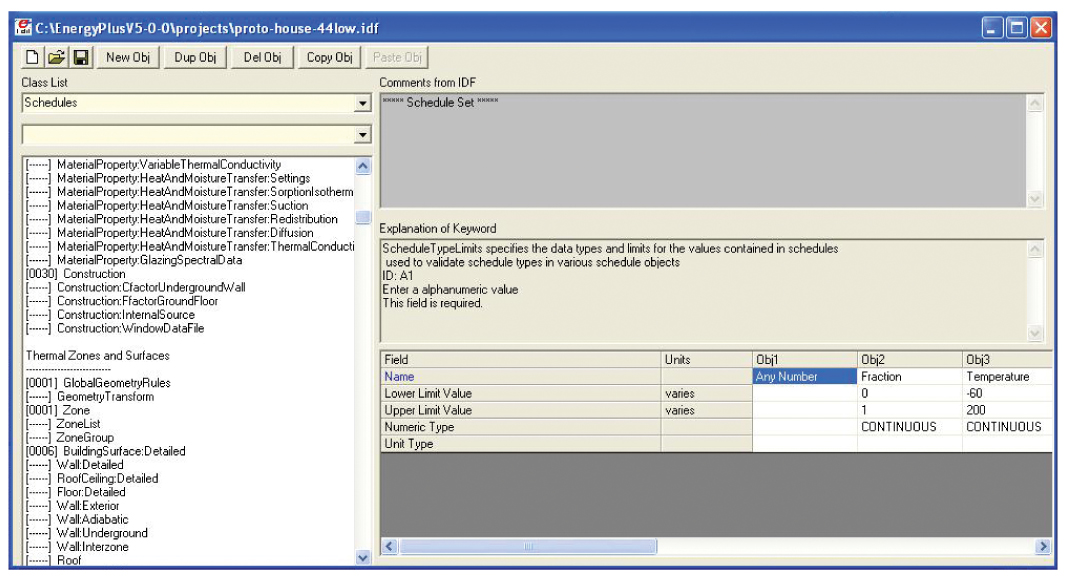


Ilustración 8. Interfaz de EnergyPlus con el archivo de extensión “.idf” del modelo del edificio abierto. (8)

* 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.

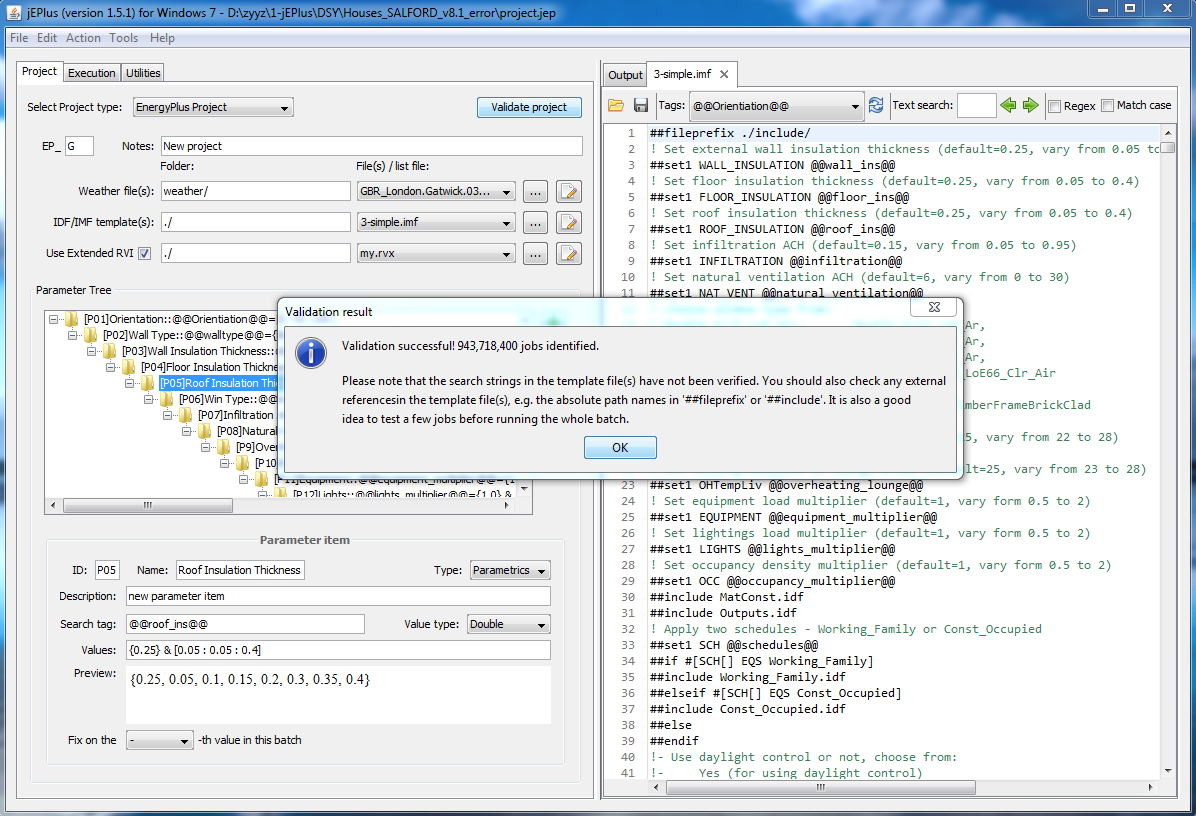


Ilustración 9. JEplus siendo usado desde su interfaz gráfica. (9)

* Adicionalmente 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 encuentra forma de recuperar su estado anterior estable o simplemente se perdiese la información almacenada, se ha utilizado “Github” como software adicional 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”.* (10)

## UML

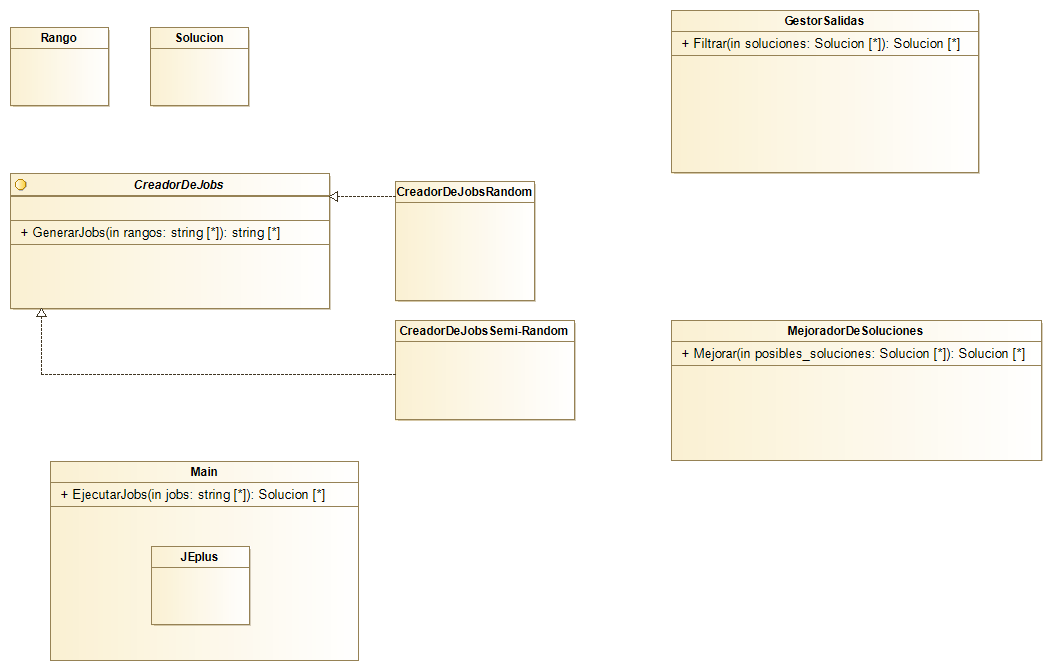


Ilustración 10. Diagrama UML de las clases creadas.

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 envían 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 “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 funciones 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 esté 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 envían 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 habiéndose 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). Además de tener que estar creando “jobs”, enviándolos a JEplus, y tener que esperar este devuelva el valor de las salidas.

* “Main”: Esta es la clase principal, su importancia reside en que abre e inicializa el proyecto de JEplus para que este pueda ejecutar los “jobs” que creemos.

Cabe destacar además que contiene una serie de constantes fundamentales para su ejecución y para poder ejecutar los métodos de las demás clases:

* “NUMERO\_JOBS”: Como indica el nombre, es el número de “jobs” que se desean crear. Se pueden generar tantos como se deseen, pero en nuestro caso hemos fijado el número en cien, ya que lo consideramos suficiente para obtener buenos resultados y si se añaden muchos más los tiempos de ejecución aumentan considerablemente.
* “NUMERO\_PARAMETROS”: Especifica el número de parámetros que se van a tener en cuenta para calcular las funciones energéticas a optimizar. Es fundamental que coincida con el número de parámetros asociados al proyecto declarados en el archivo de extensión “mvi” asociado al proyecto.
* “RANGOS\_DE\_LOS\_PARAMETROS”: Es un array de tamaño especificado por la constante “NUMERO\_PARAMETROS”. Cada elemento del array es un “Rango” cuyo valor mínimo y máximo de cada rango se obtiene del archivo de extensión “imf” asociado al proyecto.
* “ELEMENTOS\_SALIDA\_MAXIMIZAR”: Esta constante es un array de booleanos que indican si los valores de las funciones de salida especificadas en un orden en el archivo “imf” se desean maximizar o minimizar. En el caso del consumo eléctrico puede parecer que lo evidente es desear minimizarlo, pero hay tantas posibles opciones de funciones de salida como puede ser la refrigeración o humedad que no sea tan evidente y por tanto se debe especificar.
* “Ruta”: Esta constante especifica el directorio del ordenador donde se encuentra el proyecto del que se van a cargar los ficheros que contienen toda la información de parámetros y funciones a optimizar.

La clase “Main” también se encarga de crear objetos de las demás clases anteriormente mencionadas y usarlos en el orden correcto. Envía las entradas a un objeto y las salidas que produce este se envían sin modificaciones, directamente, como entrada al método principal de otra clase. El orden sería el siguiente:

En primer lugar “Main” crea un objeto de la clase “CreadorDeJobs”, al que pasa como parámetro la constante “RANGOS\_DE\_LOS\_PARAMETROS”. Con esta información es capaz de generar y de devolver a “Main” un array de “jobs”, que como ya se sabe, es a su vez un array de “strings”. Inmediatamente después “Main” llama a la librería ya configurada de JEplus y le envía como parámetros los “jobs” creados. JEplus devuelve las salidas a “Main” y este las encapsula en un array de objetos de tipo “Solucion”, y a continuación crea un objeto de la clase ”GestorSalidas” que filtra, como ya se ha explicado, las que no contienen ninguna función de salida que sea mejor que las del resto de soluciones.

En este momento “Main” puede terminar y ofrecer como soluciones las que ya se han filtrado o puede seguir el proceso e intentar mejorarlas pasando este array de “Solucion” a un objeto de la clase “MejorarSoluciones”.

El orden de llamadas a las diferentes clases se especifica a continuación en el siguiente esquema:

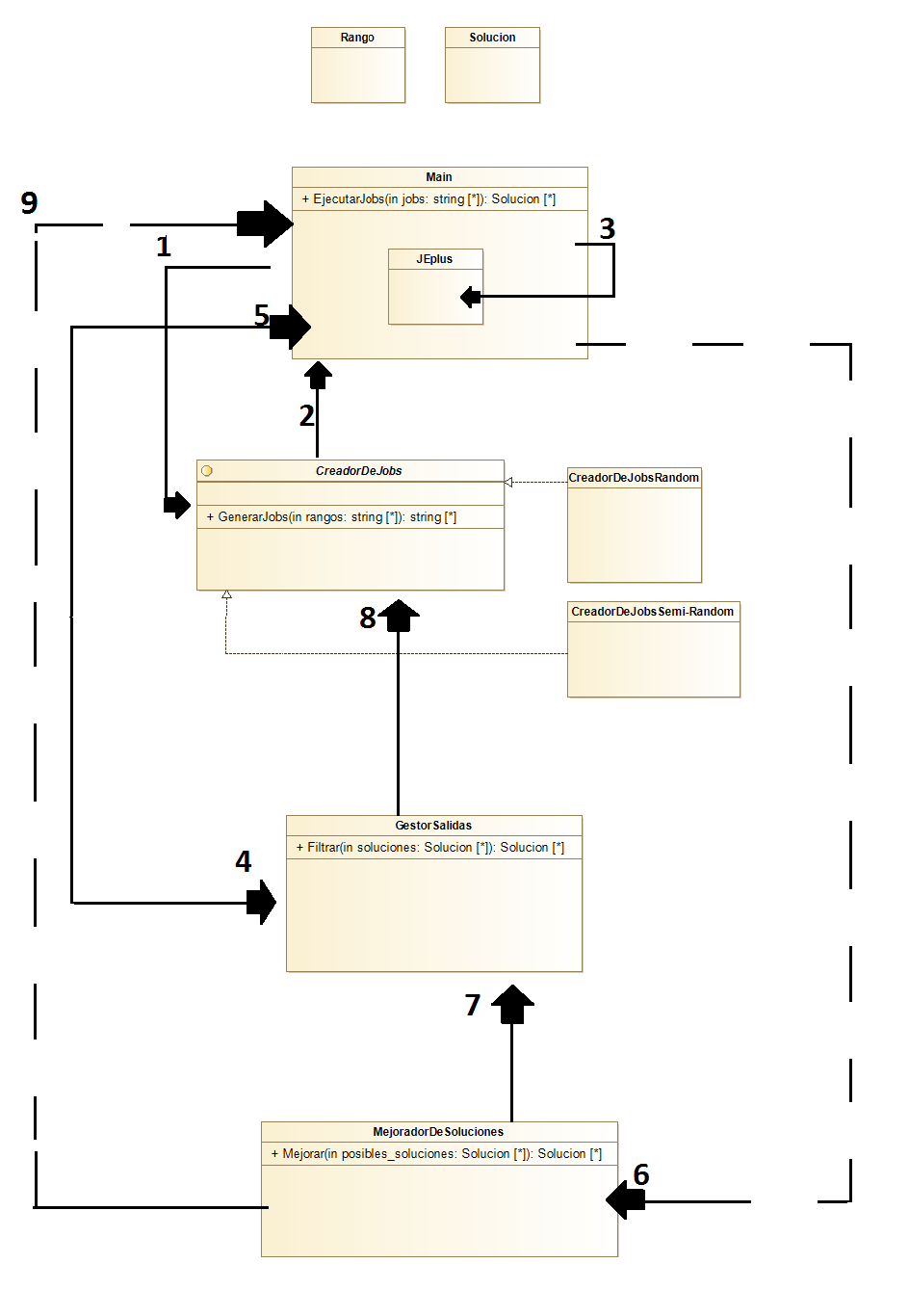


Ilustración 11. Orden de llamadas entre las clases creadas.

# Resultados

Se ha explicado todo lo relacionado con el proyecto, los objetivos que se buscan conseguir, su construcción, las partes que lo componen y su funcionamiento desde el primer paso en el que generamos “jobs” a partir de los parámetros asociados al proyecto, hasta que se devuelve al usuario unos valores para las funciones a optimizar razonablemente adecuados.

Una vez explicado cómo funciona el programa en general, se procede a ejecutar nuestro programa para un modelo de edificio concreto, con los distintos métodos de generación de “jobs” que han sido implementados y se analizan los resultados obtenidos.

El modelo de edificio empleado para la extracción de resultados se ha obtenido de la propia web de “JEplus”. Corresponde a un modelo de edificio diseñado para Chicago, Illinois, Estados Unidos.

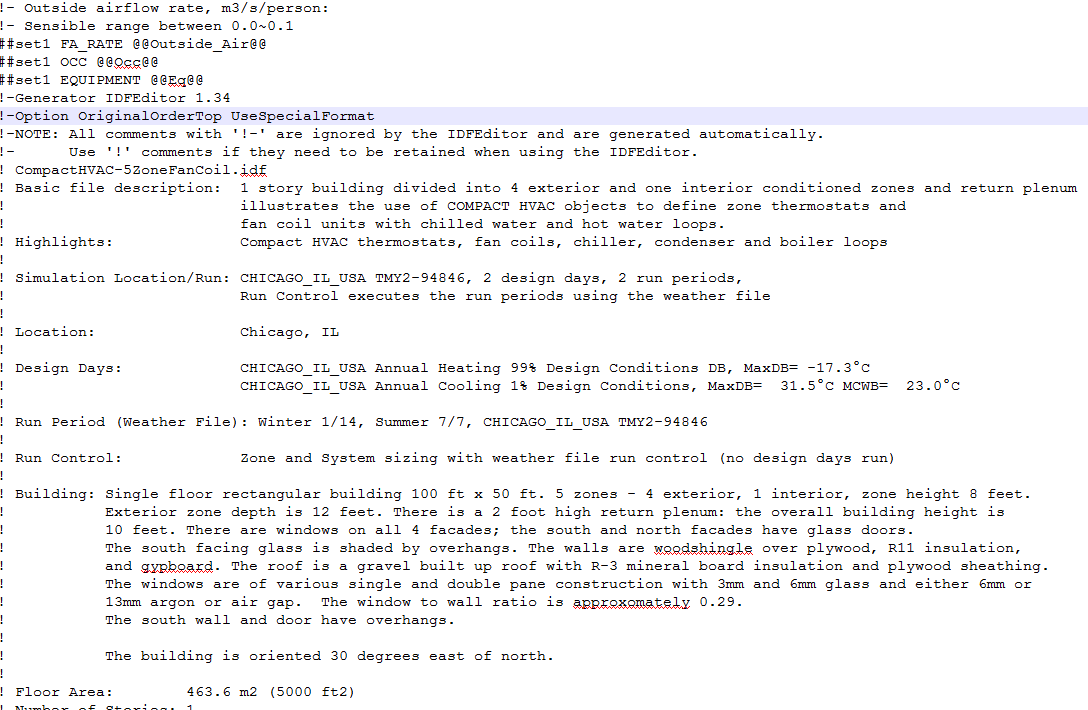


Ilustración 12. Archivo "imf" con la descripción del modelo de edificio empleado para obtener los resultados

Los archivos del proyecto se han modificado para que se consideren cuatro parámetros variables: “Orientación del edificio”, “cantidad mínima de aire puro”, “densidad de gente por metro cuadrado” y “Potencia del equipo en función de la densidad de ocupación”.

Respecto de las funciones a optimizar, se han decidido fijar tres que se han considerado interesantes y con muchos posibles valores. Estas son: el consumo de electricidad por todo el complejo del edificio, el consumo de las bombillas del interior del edificio y el gasto en gas. Se puede calcular el consumo por horas, meses, anual y el que hemos decidido, ya que estaba por defecto en el proyecto del edificio, el consumo generado entre el 14 de enero y el 7 de julio. Se busca maximizar todas las salidas. Esta decisión se corresponde únicamente a que a la hora de mostrar los resultados es más visual y sencillo.

Una vez finalizado el programa se comprueba que valores de los cuatro parámetros elegidos son los adecuados para las tres salidas elegidas.

El resto de configuraciones realizadas antes de ejecutar el programa son las siguientes:

Anteriormente se ha explicado que el factor “alpha” para discretizar el rango depende del número de “jobs” a ejecutar. Pero, para decidir esto, hubo un proceso en el que surgieron una serie de problemas.

En primer lugar, se desconocía en base a qué factor asignarlo. Se pensó como una constante. En las pruebas realizadas se comenzó fijando en 100, y se obtuvieron resultados aceptables, pero esto no era así si se aumentaba mucho el número de “Jobs”, ya que al fragmentar “tan poco” el rango, al intentar mejorar soluciones, muchas se repetían. Se aumentó entonces la constante “alpha” a 500, para probar, pero en este segundo caso, si no se creaban muchos “jobs” apenas se producían cambios en la mejora de soluciones, y todo ese proceso era muy ineficaz. Por tanto, la solución encontrada fue asignar el valor de “alpha” en base a número de “jobs” creados. Esto explica el porqué de nuestra decisión.

La siguiente configuración es, por tanto, el número de “jobs” a crear. Se ha decidido crear 200 “jobs”. Es un número lo suficientemente grande para poder filtrar muchas soluciones no optimas y para, a su vez, devolver un número adecuado de salidas válidas a comparar entre sí.

Además, se ha calculado que el tiempo aproximado de ejecución por cada “job” para generar una salida es de 18 segundos. Como se quiere repetir la ejecución del programa varias veces para estudiar la variabilidad de soluciones obtenidas, pero dentro de unos límites de tiempo razonables (suponemos 24 horas como tiempo razonable para obtener todos los resultados):



Resolviendo esta sencilla ecuación se obtiene que aproximadamente, ejecutar hasta 24 veces se realizará en menos de un día. Pero como hay que probar para nuestros dos métodos de generación de “jobs”, el “Random” y el “Semi-Random” hay que contar con el doble de tiempo y por tanto se redondea y se decide que es razonable ejecutar 10 veces el programa para estudiar las diferentes soluciones que se generan y si estas varían mucho entre sí.

Justificadas nuestras decisiones respecto la configuración de las pruebas finales a realizar, se procede a explicar los resultados.

## Experimentos

Tras 20 horas 58 minutos y 45 segundos de ejecución, con 4000 “jobs” totales creados, obtenemos 192 soluciones. Se crean 4000 en total porque creamos 200 por ejecución, por 10 ejecuciones para cada uno de los 2 métodos de generación de “jobs” creados, el “Random” y el “Semi-Random”.

Simplemente con estos datos podemos ver la ventaja de emplear un enfoque metaheurístico para obtener soluciones. Mientras que antes los usuarios de JEplus tenían que intuir que “jobs” construir o crear todas las combinaciones posibles, y luego revisar cada salida de cada “job” para ver cuál era la mejor; ahora tan solo se especifica el número de “jobs” que se quieren crear y se eliminan automáticamente las peores soluciones.

De esas 192 soluciones, 103 se generan a partir del método “Random” y 88 del “Semi-Random”. Este hecho sirve para tener un primer indicio de que el método “Semi-Random” tiene mayor diversidad en el valor de sus soluciones y permite filtrar más resultados, pero como aún desconocemos la calidad de las soluciones aportadas por cada uno, todavía no se puede decir que sea mejor.

Si se aplica la mejora de soluciones una vez obtenidas con ambos métodos, a partir de las 103 obtenidas del “Random”, se mejoran, se vuelven a filtrar y se reducen a 45 soluciones.

Si se realiza esto mismo para las 88 generadas por el método “Semi-Random”, se disminuye a 44 soluciones.

Si se elige mejorar las salidas, se obtendrían 89 posibles soluciones de las 4000 posibles que se habían creado. Esto implica que si con las configuraciones actuales (200 “jobs” por ejecución) se ejecuta una sola vez el programa, con un solo método de generación de “jobs”, se obtiene una media de 4,45 soluciones posibles. Esta estadística es fundamental ya que prueba que el número de soluciones se reduce de media en 97.7%.

A continuación, vamos a comparar entre sí la calidad de los resultados obtenidos con los diferentes métodos de generación de “jobs” y cómo cambian cuando se les aplica el proceso de mejora.

La calidad de los resultados se analiza en función a los óptimos de Pareto. Como se ha explicado en apartados anteriores, una solución es peor que otra solo si consigue peores resultados en todas sus funciones a optimizar, porque no se puede determinar un orden sobre qué función es más importante o prioritaria. Todas las soluciones que cumplan que no son mejores o peores que las demás, se encuentran en la frontera de Pareto. Se aplica su término, pero en nuestro problema:

*“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. “* (11)

## Preliminares

En las pruebas preliminares se examinan y se comparan las soluciones obtenidas por los métodos “Random” y “Semi-Random”.

Ambos conjuntos de soluciones comparten una característica, la función de salida “consumo de electricidad por el complejo del edificio”, que no varía en ninguna solución. Su valor siempre es “3.09E12”. Se puede determinar que, casualmente, se ha elegido optimizar una función de salida en la que no influyen los parámetros variables elegidos para esta recolección de resultados. Es interesante comprobar y tiene sentido que no siempre todos los parámetros van a afectar a todas las salidas. Esto además nos facilita mostrar en una imagen los resultados obtenidos, ya que, como solo varían dos de las funciones a optimizar, se pueden mostrar todas las soluciones en un gráfico 2D.

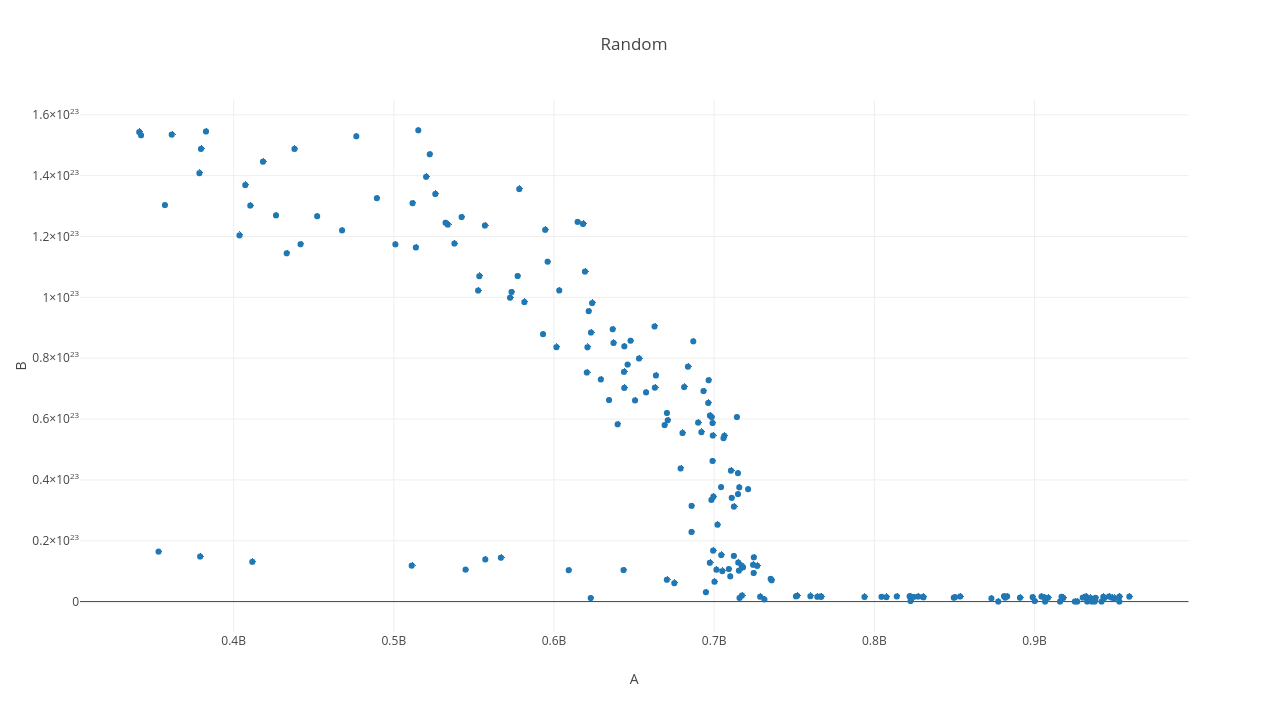


Ilustración 13. Gráfico de las 103 soluciones obtenidas por el método "Random".

Es importante aclarar que hay soluciones que se encuentran por debajo de los optimales de Pareto porque se debe a otra ejecución distinta. En este gráfico se encuentran las salidas de todas las ejecuciones totales (10), pero para cada ejecución, todas aquellas salidas que se encuentren en la zona inferior de los optimales de Pareto son filtradas.

Se observa además la tendencia de resultados, realizando una frontera de Pareto evidente de las dos funciones a optimizar. Esto es, una curva de soluciones cuyo límite superior es la solución cuya función de salida “Consumo de gas” tiene el valor más alto, que en este caso es “1.54509E+23”. La otra parte de la curva viene definida por la solución cuya otra salida, “Consumo de las bombillas del interior del edificio” tenga el mayor valor, en este caso es “9.52981897021429E8”.

A continuación, se crea y se analiza la gráfica a partir de los resultados obtenidos con el método “Semi-Random”.

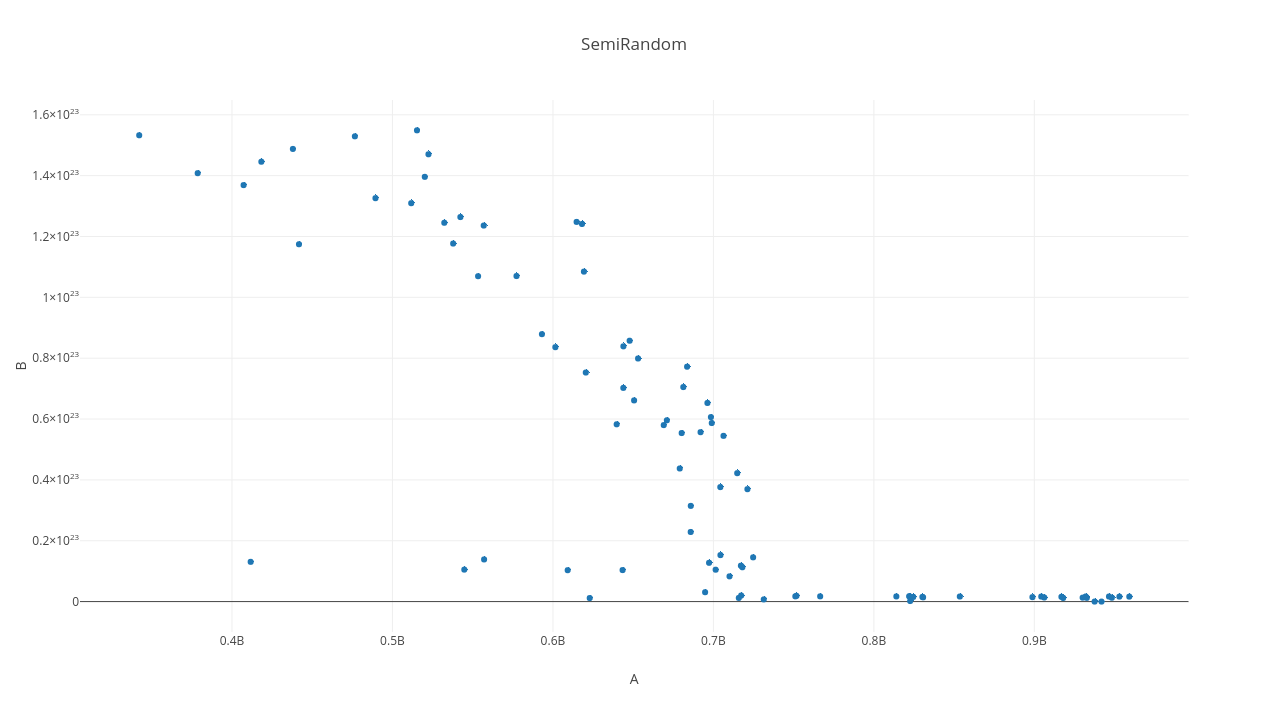


Ilustración 14. Gráfico de las 88 soluciones obtenidas por el método "Semi-Random".

Se comprueba como la curva generada es muy similar al caso anterior. Con en el método “Semi-Random” la frontera de Pareto está definida por el mayor valor de la función “Consumo de gas”, que en este caso es “1.5489E23”, y por la salida con mejor valor en la función “Consumo de las bombillas del interior del edificio”, en este caso es “9.59168695474667E8”.

Valores ligeramente mejores que los dos que definían la frontera de Pareto generada con el método “Random”. Además, con este método se genera una menor densidad de soluciones posibles.

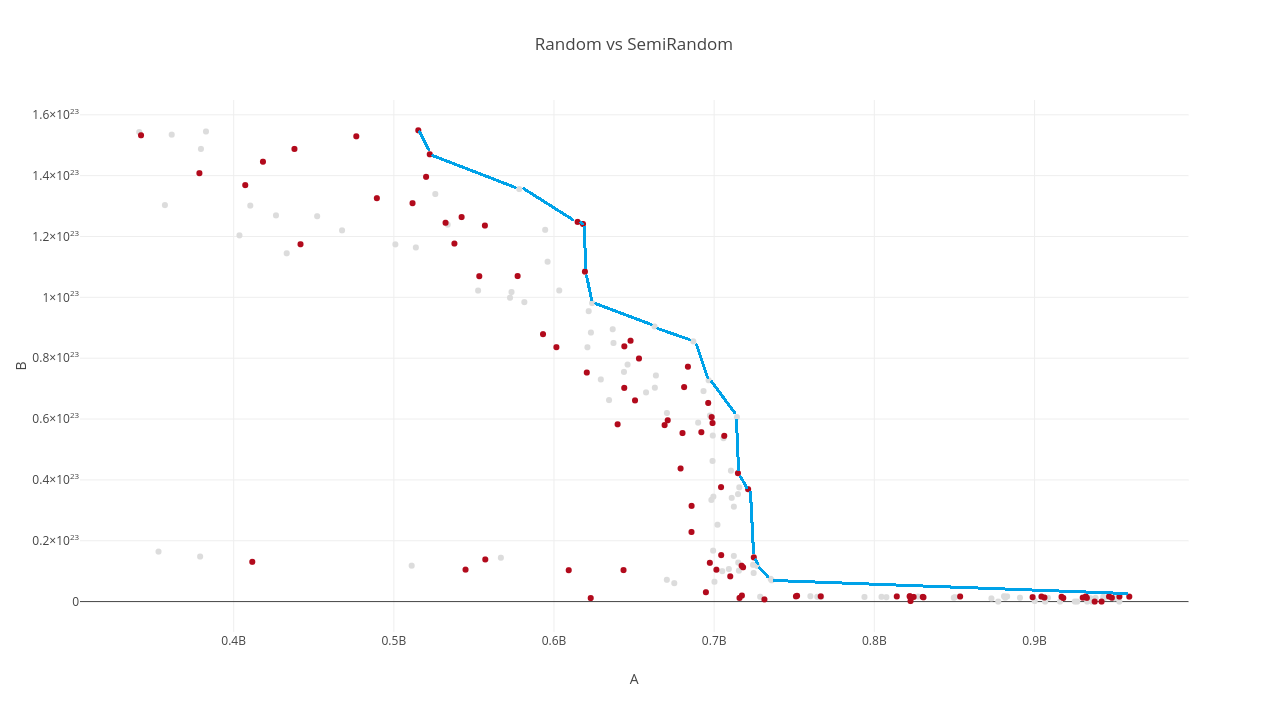


Ilustración 15. Comparación de las soluciones generadas con el método "Random" y "Semi-Random".

En la gráfica anterior se combinan las soluciones generadas con el método "Random" (Gris) y "Semi-Random" (Rojo).

Se dibuja una línea definiendo la frontera de Pareto con los mejores valores de ambos conjuntos de soluciones. La frontera se define con un total de 17 soluciones, 9 de ellas pertencienetes a las generadas con el método “Semi-Random” y 8 por el método de construción de jobs “Random”.

Se puede decir, por tanto, que la calidad de las salidas generadas por ambos métodos está muy equilibrada, aunque ligeramente mejor el segundo caso, que además genera menor densidad de soluciones. Con todo esto se puede decir que el caso “Semi-Random” es mejor, aunque a continuación se comprueba si esto cambia cuando las soluciones obtenidas hasta el momento se intentan mejorar.

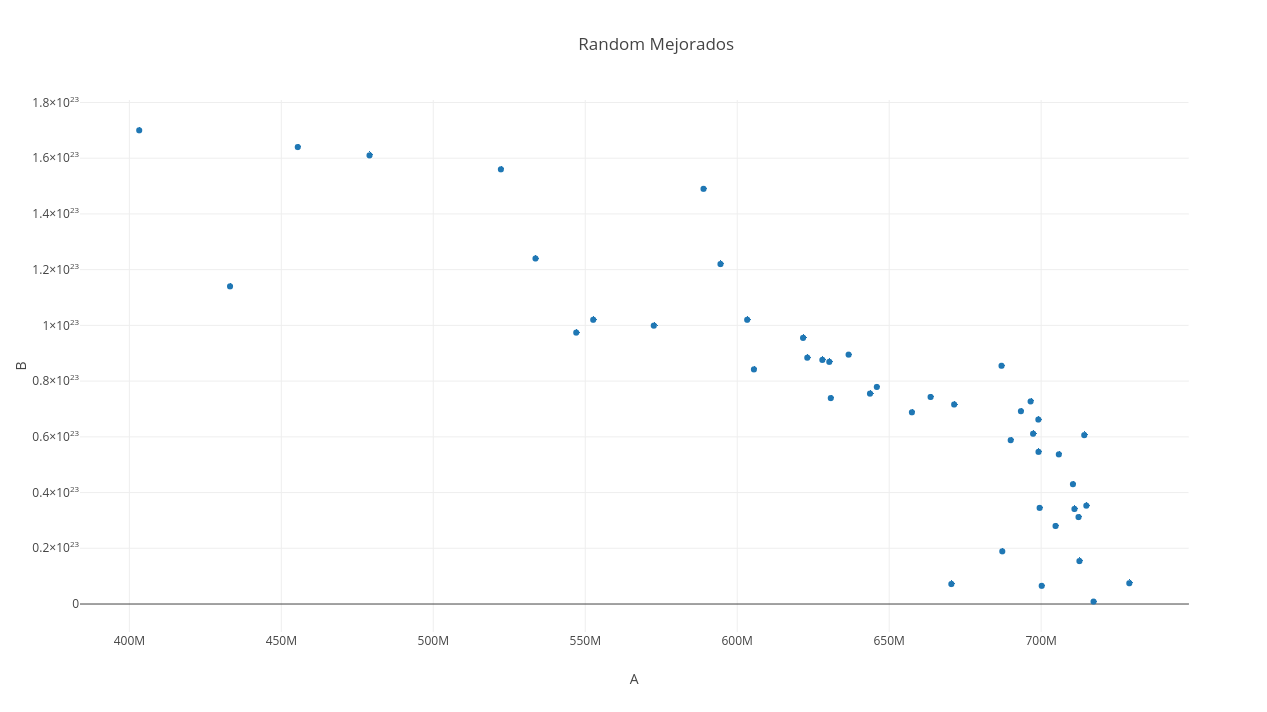


Ilustración 16. Gráfico de las 45 soluciones obtenidas al mejorar las obtenidas por el método "Random".

Al mejorar las soluciones obtenidas por el método “Random” se reduce la densidad de soluciones. Visualmente se comprueba que muchas de las soluciones que se encontraban por debajo de la frontera de Pareto desaparecen.

Los valores que definen la frontera también aumentan. El mayor valor del “Consumo de gas” crece hasta “1.70E23”, a costa de disminuir el “Consumo de las bombillas del interior del edificio” cuyo mayor valor pasa a ser “7.29003349409338E8”. Este valor se reduce porque si aplicamos la mejora de soluciones, en el caso de no conseguir mejorar una solución, pero se encuentra una alternativa igualmente buena, guardamos esta nueva. Aunque se puede decidir también guardar la original.

Otra consecuencia de la mejora de soluciones es, aparte de reducir el número de soluciones a 45, la frontera de Pareto tiene menor curvatura, esto implica que se aprovecha más el conjunto de los posibles los valores optimales.

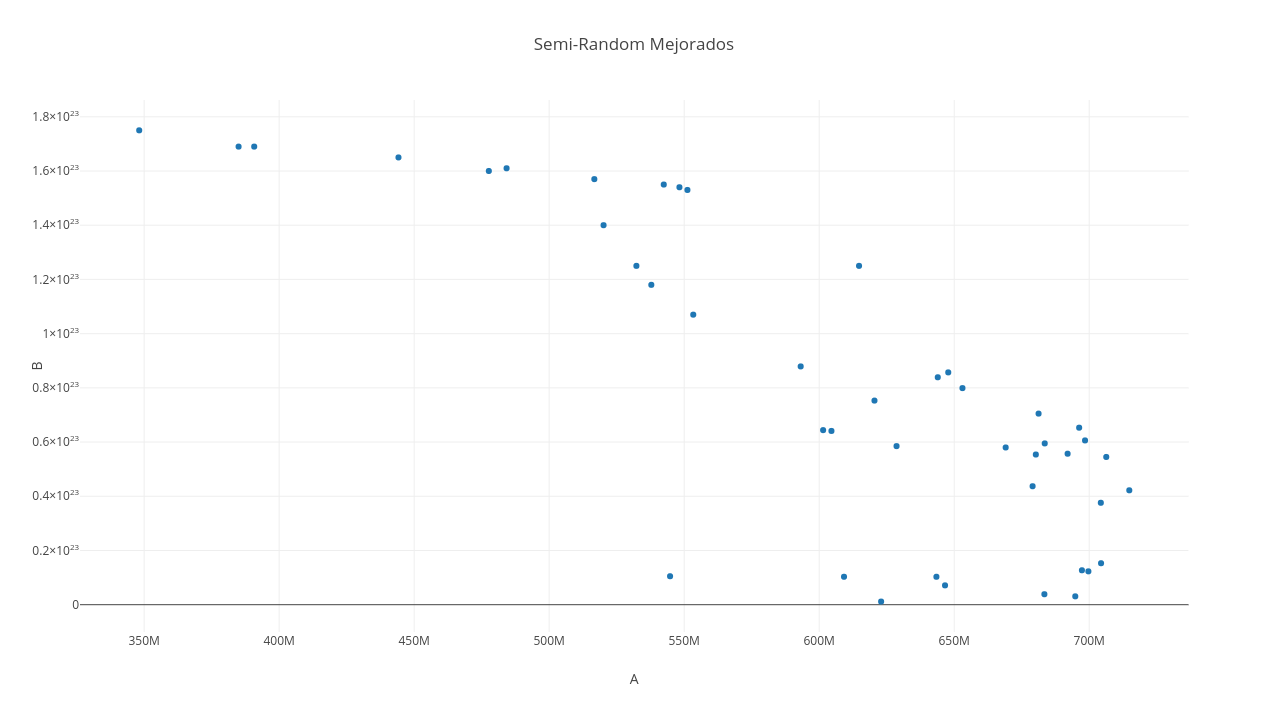


Ilustración 17.Gráfico de las 44 soluciones obtenidas al mejorar las obtenidas por el método "Semi-Random".

De manera similar, al mejorar las soluciones obtenidas por el método “Semi-Random” comprobamos que se reduce la densidad de soluciones a 44. Se desplazan en el gráfico hacia la derecha y hacia arriba muchos valores, que se traduce en que muchas soluciones han mejorado sus valores tanto en el “Consumo de gas” como en el “Consumo de las bombillas del interior del edificio”.

## Finales

A continuación, se comparan los resultados finales, los obtenidos mediante las soluciones mejoradas obtenidas por ambos métodos de generación de “jobs”, el “Random” y el “Semi-Random”. El objetivo es averiguar cuál de ellos de se beneficia más de este proceso de mejora, y así, determinar finalmente que opción presenta mejores resultados.

Se forma la gráfica correspondiente para cada tipo de solución, se combinan y se comparan resultados:

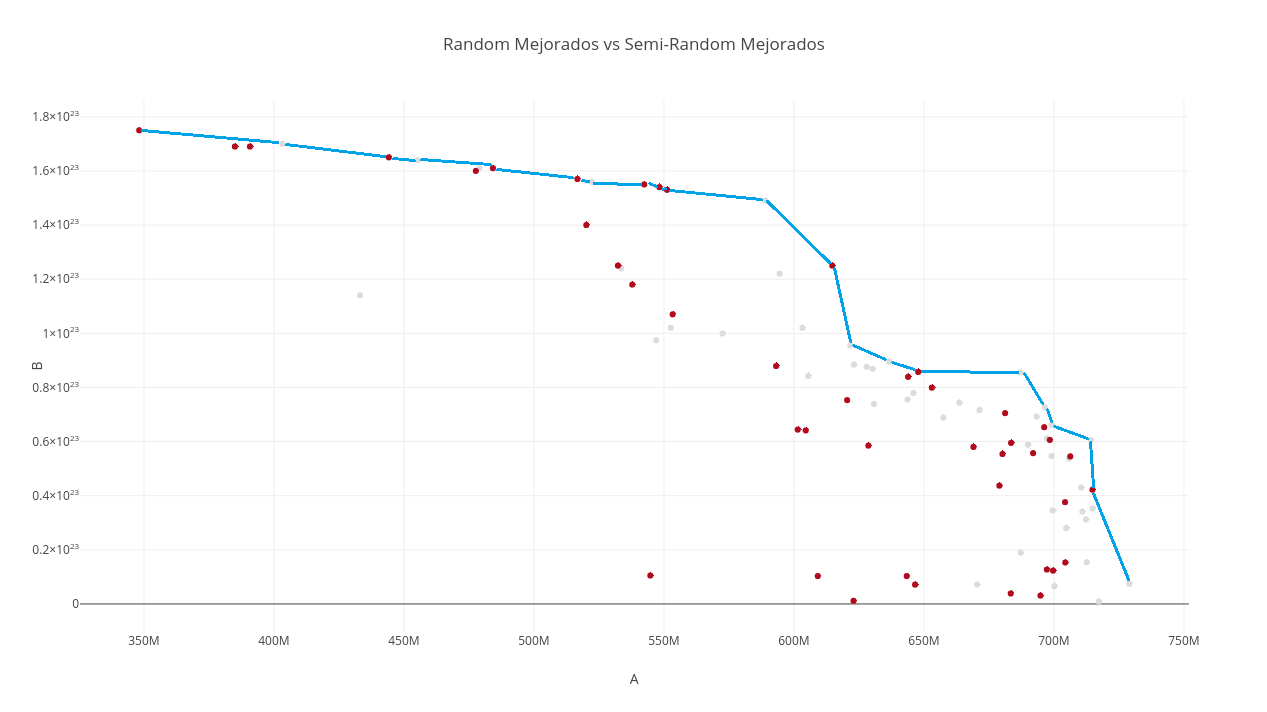


Ilustración 18. Comparación de las soluciones mejoradas generadas con el método "Random" y "Semi-Random".

Comparando la información presente en el gráfico, se concluye que el número de soluciones obtenidas por ambos métodos es similar, 45 del “Random” (Color gris) y 44 del “Semi-Random” (Color rojo). Con lo estudiado en los primeros análisis de resultados, sabemos que el “Random” ha tenido que filtrar de 103 a 45 y el “Semi-Random” de 88 a 44. En este punto es preferible utilizar el método “Semi-Random” ya que el tiempo de ejecución se reduce al no tener que realizar el filtrado y la mejora para una cantidad alta de soluciones que luego deberán eliminarse como ocurre en el caso del método “Random”. Se puede decir que el “Semi-Random” encuentra soluciones más “inteligentes” y diversas.

El siguiente aspecto a analizar es la calidad de ambos tipos de soluciones, para ello dibujamos la frontera de Pareto formada por las mejores soluciones de ambos tipos. La frontera se forma con 21 soluciones, 10 formadas con el método “Semi-Random” y 11 pertenecientes al “Random”. El resto de valores se encuentran muy cerca de la frontera y entre sí, con pequeñas diferencias en el valor de las funciones a optimizar.

La conclusión respecto la calidad no es contundente, pero en este caso, para las pruebas realizadas con este modelo de edificio y las configuraciones actuales, las soluciones obtenidas por el método “Random” son, aunque muy ligeramente, mejores.

Como resumen final de todos los resultados y comparaciones realizadas concluimos:

* El tiempo de ejecución de nuestro programa empleando el método “Semi-Random” es algo menor. Es más eficiente porque encuentra en menos tiempo un número similar de soluciones, pero dado que el tiempo de ejecución no es un factor demasiado relevante para este programa tampoco se puede decantar en base a esto por una u otra opción.
* Como se ha comprobado en la comparativa entre ambos métodos también la calidad de las soluciones es similar. Las soluciones proporcionadas por el método “Random” son ligeramente mejores, pero la diferencia es tan ínfima que se debería ejecutar el programa un número muy elevado de veces, y con distintos modelos de edificios y distintas configuraciones para poder determinar si de forma general un método es superior frente a otro.

* Por tanto, y uniendo las dos conclusiones anteriores, en la que mediante el método “Semi-Random” se obtiene un tiempo de ejecución ligeramente mejor, y que mediante el método “Random” se obtienen soluciones ínfimamente mejores. Es razonable concluir que no se puede determinar de forma general cuál de los dos métodos reporta mejores soluciones. Es necesario analizar cada problema de cada modelo de edificio de forma particular, y así ejecutar, o ambos métodos o utilizar el que haya sido probado como mejor para ese problema concreto.

Aunque para un problema de optimización energética general no se pueda asegurar que método de generación de “jobs” es mejor, sí se puede concluir que ambos métodos devuelven soluciones razonablemente válidas y cercanas a su optimalidad.

# Conclusiones y trayectorias futuras

Una vez realizado el proyecto y como últimas conclusiones se puede decir que, el objetivo fundamental del proyecto que consiste en optimizar cada una de las funciones de salida relacionadas con el consumo energético del proyecto, ha sido realizado con éxito. No solo reduciendo drásticamente la cantidad de resultados a considerar, si no ofreciendo únicamente los mejores.

Además de todo esto, se facilita la labor que debe realizar el usuario. Aunque disponga la opción de decidir modificar las configuraciones del proyecto: Número de parámetros variables a considerar, modificar sus rangos, funciones de salida a optimizar…, puede simplemente limitarse a especificar el número de “jobs” a crear y ejecutar el programa.

Respecto a las posibles mejoras o decisiones que se podrían haber tomado respecto el proyecto, decir que nuestro programa creado no solo es una solución al problema de tener que elegir una buena salida entre un conjunto enorme de posibilidades, si no que sirve y se ha creado con la idea de que sea una base escalable en la que se puedan añadir todos los métodos de creación de “jobs” que se deseen y todos los sistemas de filtrado de soluciones que puedan interesar a los actuales usuarios de JEplus.

Es posible que en un futuro las funciones de salida lleven asociados un peso, o que a los parámetros variables se les añada un orden de prioridad. Nuestro programa actualmente no permite esas características, pero sí está modularizado de tal manera que se pueda añadir funcionalidad que recoja estos y otras posibles opciones futuras.

# Bibliografía

1. **EnergyPlus.** EnergyPlus. [Online] https://energyplus.net/.

2. **JEPlus.** JEPlus.org. [Online] http://www.jeplus.org/wiki/lib/exe/fetch.php?w=750&tok=68280d&media=jeplus\_v1.5\_diagrams.png.

3. **Instituto Nacional de Astrofísica, Óptica y Electrónica de México.** Instituto Nacional de Astrofísica, Óptica y Electrónica. [Online] https://ccc.inaoep.mx/~emorales/Cursos/Busqueda/node58.html.

4. **Galiano, Fernando Berzal.** Fernando Berzal. [Online] http://elvex.ugr.es/software/nc/help/spanish/nc/clustering/GRASP.html.

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

5. **GRAFO.** grafo RESEARCH. [Online] http://www.grafo.etsii.urjc.es/sites/default/files/papers/CAEPIA%20%282%29\_3.pdf.

6. **Fernández-Viagas Escudero, Víctor.** e-REdING. Biblioteca de la Escuela Superior de Ingenieros de Sevilla. [Online]

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

7. **Shahnazari-ShahrezaeiI, P., et al.** SciELO Scientific Electronic Library Online. [Online] 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.

8. **Home Power Inc.** Home Power Magazine. [Online] https://www.homepower.com/sites/default/files/uploads/Smith141HEED5.jpg.

9. **Building Energy Software Tools.** Best Directory. [Online] https://d2awrjepde3yht.cloudfront.net/files/styles/img\_280\_210/public/jeplus\_screenshot.png?itok=tHbVNc6p.

10. **Plotly.** Plotly. [Online] https://plot.ly/.

11. **Pareto, Vilfredo.** *Cours d’Économie Politique.* 1896.