2. METODOLOGÍA.

2.1. Optimización empleando GenOpt y un simulador térmico de edificios.

Cuando el objetivo es mejorar el diseño del edificio para reducir la demanda energética, las actuaciones que se deben llevar a cabo en líneas general es son:

En invierno:

- Diseño urbano favorable (acceso solar).
- Limitar pérdidas (aislamiento).
- Promover ganancias (orientación, superficie de vidrios, inercia).

En verano:

- Diseño urbano favorable (control solar, microclima).
- Limitar ganancias (control solar, modulación).
- Promover pérdidas (ventilación).

El objetivo de la optimización que pretendemos realizar es, partiendo de un diseño inicial de la envuelta, obtener un diseño final mejor desde el punto de vista de una determinada función objetivo, que podrían ser por ejemplo el consumo energético total del edificio o el coste total en materiales más el consumo energético a lo largo de toda la vida útil del edificio.

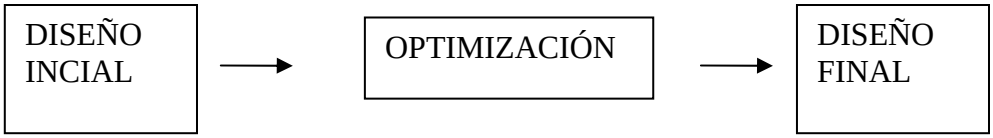


Fig. 4: Optimización del diseño inicial

Además una mejora en el diseño de la envolvente del edificio puede suponer una mejora en la certificación energética del mismo. En este aspecto un estudio económico podría resultar interesante para la toma de decisión en la elección de un diseño u otro.

Teniendo en cuenta todo lo anterior, nos preguntamos si es posible obtener esta optimización del diseño haciendo trabajar conjuntamente al programa de simulación térmica de edificios desarrollado por el Grupo de Termotecnia de la Escuela Superior de Ingenieros de Sevilla de cálculo de la energía demandada en base horaria en calefacción y refrigeración por un edificio con el programa GenOpt de optimización genérica de una determinada función objetivo con sus correspondientes parámetros libres sujetos a restricciones. Y si es posible esto, qué tipos de complejidad de casos podrían ser empleados con éxito y en cuánto tiempo podría llevarse a cabo el análisis.

Pero antes de nada, conozcamos más acerca de los programas GenOpt y simulador térmico de edificios, así como de algunas de las particularidades del problema de optimización al que nos vamos a enfrentar.

2.2. GenOpt.

2.2.1. Introducción.

GenOpt, (Generic Optimization Program copyright (c) 1998-2003, desarrollado por la Universidad de California, a través del Lawrence Berkeley National Laboratory), es un programa genérico de optimización para la minimización de una función objetivo multidimensional que es evaluada mediante otro programa externo.

Ha sido creado para problemas de optimización donde la función de coste es computacionalmente compleja y sus derivadas no puedan ser calculadas o ni siquiera existan. No ha sido diseñado para problemas de programación lineal, cuadrática o aquellos donde el gradiente de la función de coste esté disponible, ya que para éstos tipos de problemas existen otros procedimientos más eficaces. GenOpt puede acoplarse, en principio, a cualquier programa de simulación externo que lea sus entradas (valores de las variables independientes) de uno o más archivos de texto y escriba su salida (es decir, el valor de la función objetivo) a un archivo de texto. Este programa externo es llamado iterativamente (hasta que se produzca la condición de parada) por GenOpt, que genera automáticamente los valores de los parámetros libres seleccionados (las variables independientes) que minimizan la función objetivo.

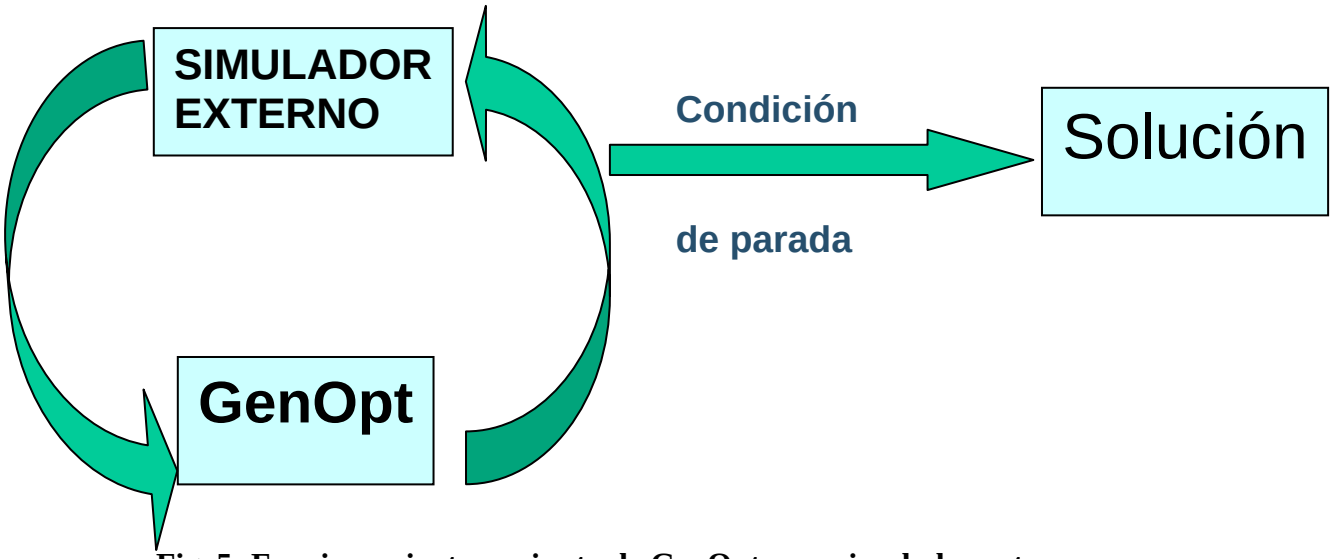


Fig. 5: Funcionamiento conjunto de GenOpt y un simulador externo.

Simulador externo.
Calcula el valor de la función objetivo.

- Lee el archivo de texto con el nuevo conjunto de valores generado por GenOpt.
- Escribe un archivo de texto con el valor de la función objetivo.

GenOpt.

- Genera el nuevo conjunto de valores de los parámetros libres.
- Escribe un archivo de texto con el nuevo conjunto de valores

a calcular por el simulador externo, en base al algoritmo seleccionado.

- Lee de un archivo de texto el valor de la función objetivo,

calculada a partir de los resultados del simulador externo.

Las variables independientes pueden ser continuas (con límites inferior y superior), discretas o ambas. GenOpt posee una librería con algoritmos para la optimización de problemas unidimensionales y multidimensionales. Además el usuario puede implementar su propio algoritmo si así lo desea. Está programado en Java para garantizar así su independencia y poseer así una interfaz general que garantiza una gran versatilidad en cuanto a su aplicación en un amplio campo de problemas de optimización.

El principal campo de aplicación está en la simulación térmica de edificios; de hecho empleando GenOpt ya se han resuelto con éxito algunos problemas como por ejemplo la minimización del consumo de energía en un edificio de oficinas, empleando el programa EnergyPlus como simulador externo. GenOpt está diseñado para encontrar el mínimo de una función objetivo a crear por el usuario, como por ejemplo podría ser el consumo anual de energía en climatización en un edificio.

A continuación mostraremos la interfaz que GenOpt proporciona cuando solucionamos un ejemplo sencillo de optimización de una función objetivo f(x) que depende sólo de dos variables independientes: x1 y x2.

Previamente hay que definir las variables independientes y el método de optimización que queremos utilizar en el archivo Command.txt.

Así en este caso definimos:

```
Parameter
        {
                Name    = x1;
                Min     = -1000.0; (valor mínimo de la variable x1).
                Ini     = 500.0; (valor inicial de la variable x1).
                Max     = 1000.0; (valor máximo de la variable x1).
                Step    = 10; (paso).
        }
        Parameter
        {
                Name    = x2;
                Min     = -1000.0; (valor mínimo de la variable x2).
                Ini     = -500.0; (valor inicial de la variable x2).
                Max     = 1000.0 ; (valor máximo de la variable x2).
                Step    = 10; (paso).
        }
```

La función objetivo que hemos elegido para este ejemplo sencillo es:

$$y = x1*x1*x2*x2 + x2*x2 + x1*x1 + 100$$

cuyo mínimo es conocido de antemano e igual a 100 para el punto

x1=x2=0.

El algoritmo elegido es el de Nelder and Mead con la modificación de O'Neall.

Ejecutando el programa obtenemos por pantalla lo siguiente:

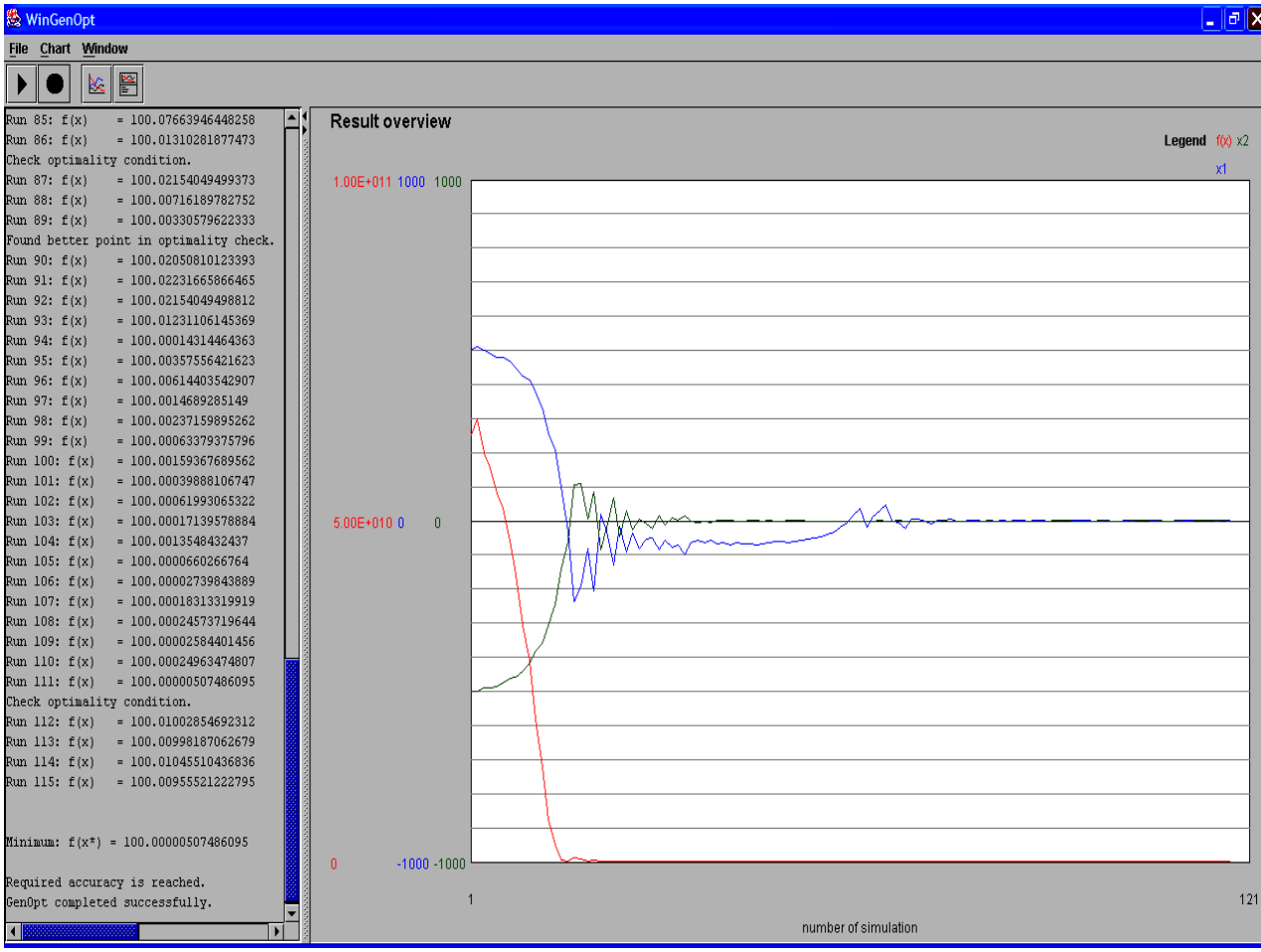


Fig.6 : Aspecto de la interfaz dada por GenOpt.

En dicho gráfico puede apreciarse la variación de los valores de ambas variables independientes (x1 en color azul y x2 en color verde) y el valor de la función objetivo (color rojo) en su camino hacia el punto óptimo.

Observamos como al cabo de 115 cálculos se cumple la condición de parada y el proceso de búsqueda se detiene, obteniéndose un valor mínimo de la función objetivo de 100.00000507486095.

Además se generan automáticamente dos archivos de texto:

- OutputListingAll.txt donde quedan recogidos numéricamente los valores de todas las variables independientes y de la función objetivo todos los pasos intermedios que ha calculado el algoritmo. Además da información acerca del operador que realiza el algoritmo en cada paso.

- OutputListingMain.txt donde quedan recogidos numéricamente los valores de todas las variables independientes y de la función objetivo exclusivamente para los puntos más destacables. Además da información acerca del operador que realiza el algoritmo en cada paso.

Consultando en cualquiera de los dos archivos citados anteriormente podemos consultar, por ejemplo, los valores en el punto óptimo de las variables independientes. En este caso obtenemos como cabía esperar:

x1=1.1668966476463538E-4 , x2=0.002249720961344792

2.2.2. Optimización.

Un problema de optimización básicamente consiste en:

1. Una serie de parámetros libres (también llamados variables independientes o parámetros de diseño).
2. Algunas restricciones que acotan el dominio de las variables independientes y dependientes.
3. Una función objetivo (a minimizar o maximizar) que depende de las variables independientes.

Ningún algoritmo de optimización trabaja mejor con todas los posibles tipos de problemas, es decir, dependiendo de la naturaleza de éste será mas conveniente un algoritmo u otro. La selección del algoritmo de optimización depende primordialmente de las siguientes consideraciones:

- La estructura de la función objetivo (linear, no-lineal, convexa, continua, número de óptimos locales etc.).
- Disponibilidad de las derivadas primera y segunda analíticamente.
- El número de variables independientes.
- Restricciones del problema en sus variables independientes y/o dependientes.

Además ningún algoritmo de optimización puede garantizar encontrar el óptimo global si existe algún otro óptimo local, ya que en la búsqueda del óptimo global el algoritmo podría quedar “atrapado” en ese mínimo local. No obstante, a pesar de este inconveniente, es obvio que es mejor encontrar esa solución aunque no sea la óptima global que no hacer nada en absoluto.

Atendiendo a estas consideraciones escogeremos cuidadosamente un algoritmo de optimización que resulte apropiado para nuestro problema.

El potencial que ofrece la simulación por ordenador no es aprovechado al máximo en la mayoría de ocasiones. Esto es generalmente debido a que las complejas interacciones entre las variables del problema en cuestión hace que los usuarios no sepan a ciencia cierta cómo elegir los valores de estas variables de entrada para obtener la solución óptima del sistema.

En ocasiones el usuario puede tardar mucho tiempo en elegir unos valores para los parámetros de entrada aceptables en el modelo de simulación. Pero una vez hecho esto, por norma general no obtendrá los valores de los parámetros que conducen al óptimo; bien sea por no disponer de tiempo suficiente para hacer el tedioso trabajo de cambiar los valores de entrada, volver a ejecutar el programa de simulación, interpretar los resultados y volver a pensar cómo mejorar estos resultados con nuevos valores de entrada para la próxima iteración o bien sea simplemente por la imposibilidad del usuario para entender todas las interacciones no lineales entre las variables de estudio debido a la complejidad del problema.

Como veremos a continuación, empleando GenOpt todo este esfuerzo puede ser aliviado, ya que mediante técnicas de búsqueda de programación matemática es posible encontrar automáticamente la solución de un problema de optimización de una o más variables independientes. Así habremos logrado el objetivo deseado: optimizar el diseño.

Para ello, GenOpt incluye varios algoritmos de búsqueda del óptimo, como son entre otros: el algoritmo de búsqueda por coordenadas, el algoritmo de Hook-Jeeves, el algoritmo multistart GPS, el algoritmo Particle Swarm Optimization o el algoritmo Simplex de Melder y Mead con la extensión de O'Neill.

Adicionalmente, GenOpt da la posibilidad al usuario para que pueda crear su propio algoritmo si así lo desea.

De entre todos los algoritmos que GenOpt incluye, elegiremos para resolver nuestro problema de optimización el algoritmo Simplex de Melder y Mead con la extensión de O'Neill, ya que es aplicable a problemas con gran número de variables independientes y además puede utilizarse para variables de tipo continuo como las que utilizaremos en nuestros casos de estudio y el algoritmo Particle Swarm Optimization, válido para variables discretas y continuas. A continuación explicaremos brevemente algo más acerca de estos algoritmos seleccionados.

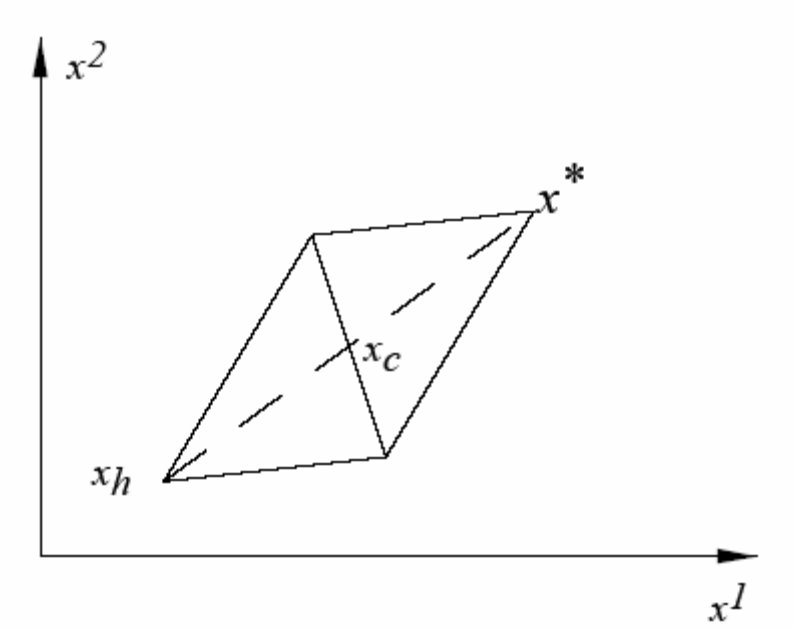
2.2.3. Método simplex de Nelder y Mead con la extensión de O'Neill.

El método simplex de Nelder y Mead se basa en la comparación directa de los valores de la función objetivo sin usar derivadas. Puede usarse para buscar la solución de un problema genérico de la optimización de una función objetivo que depende de una serie de parámetros libres sujetos a una serie de restricciones.

El algoritmo impone un simplex n-dimensional en el espacio que es ocupado por parámetros libes (más de uno). En cada uno de los n+1 vértices del simplex, el valor de la función objetivo es evaluado. En cada paso de iteración el punto con el mayor valor de la función obetivo se sustituye por otro punto.

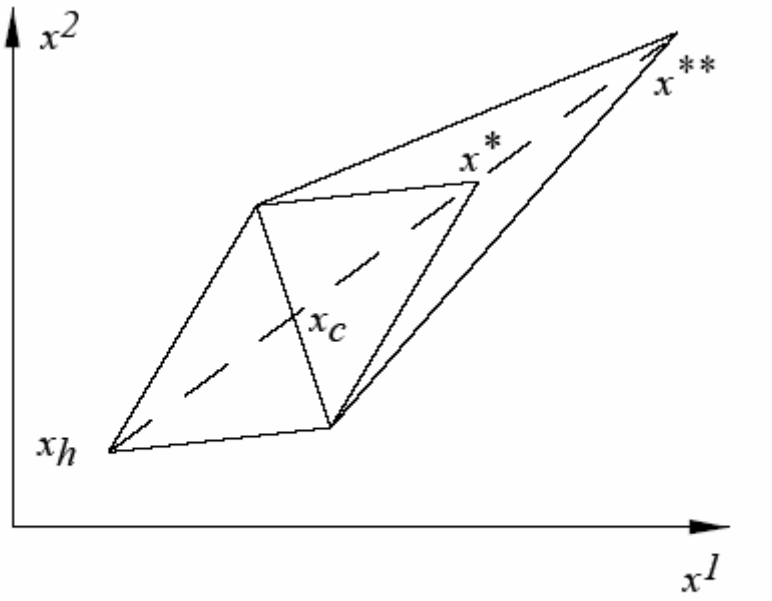
El algoritmo consta de tres operadores principales, cuya interpretación geométrica para el caso de dos parámetros libres (x1 y x2) mostramos a continuación:

a) Reflexión de un punto.



(a) Reflection

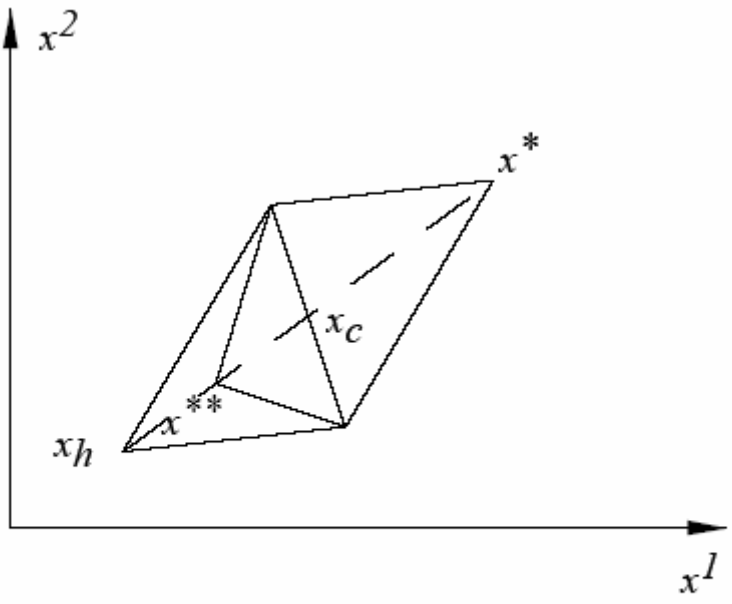
b) Expansión del simplex.



(b) Expansion

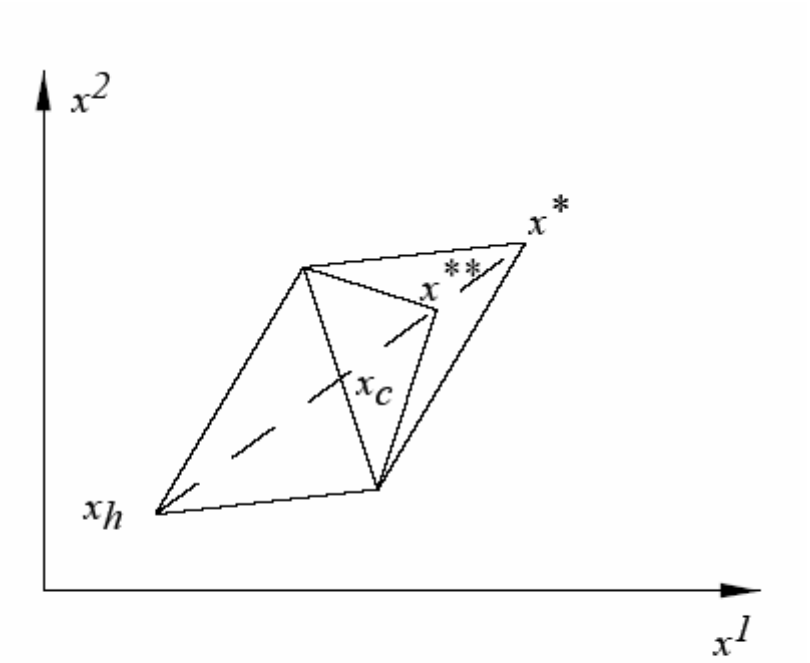
c) Contracción del simplex.

1. Contracción parcial interior.



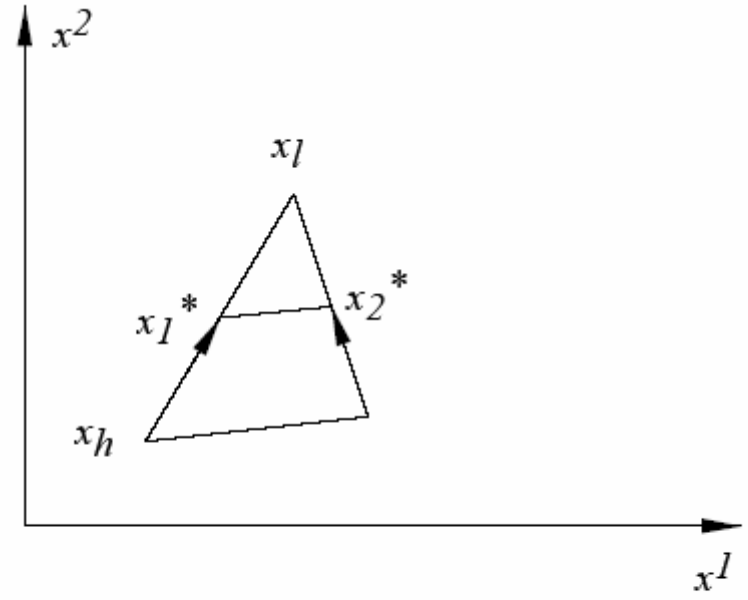
(c) Partial inside contraction

2. Contracción parcial exterior.



(d) Partial outside contraction

3. Contracción total.



(e) Total contraction

Es sabido que le método simplex puede fallar convergiendo a un punto estacionario, incluso si la función objetivo es suave o también cuando el simplex colapsa en el interior de un subespacio, o su forma se vuelve extremadamente

alongada o distorsionada. Sin embargo, a pesar de sus malas propiedades de convergencia, el método simplex a menudo localiza con éxito y con muy pocas evaluaciones de la función objetivo soluciones mejores que otros métodos de búsqueda alternativos.

En la siguiente figura observamos como avanza el algoritmo para un caso bidimensional en su camino al óptimo (ver Fig. 7).

El algoritmo se inicia definiendo en primer lugar un punto (ver punto 2), a partir del cual y con un determinado paso se crean otros 2 (ver puntos 1 y 3). A partir de aquí se calcula el valor de la función objetivo para los tres puntos y el que resulte peor de los tres (ver punto 1)es eliminado y se reemplaza por uno nuevo. Este nuevo punto se genera reflejando el punto eliminado con respecto a la recta que pasa por el segmento que forman los otros dos puntos (ver punto 4). En caso de que el valor de la función objetivo de este nuevo punto sea el mejor hasta el momento, el simplex se expande según una determinada cantidad (ver punto 5). A partir de aquí el proceso se va repitiendo iterativamente. En caso de que el valor de la función objetivo del nuevo punto reflejado sea peor a alguno de los anteriores (ver punto 12) se realizará una contracción del simplex (ver punto 13).

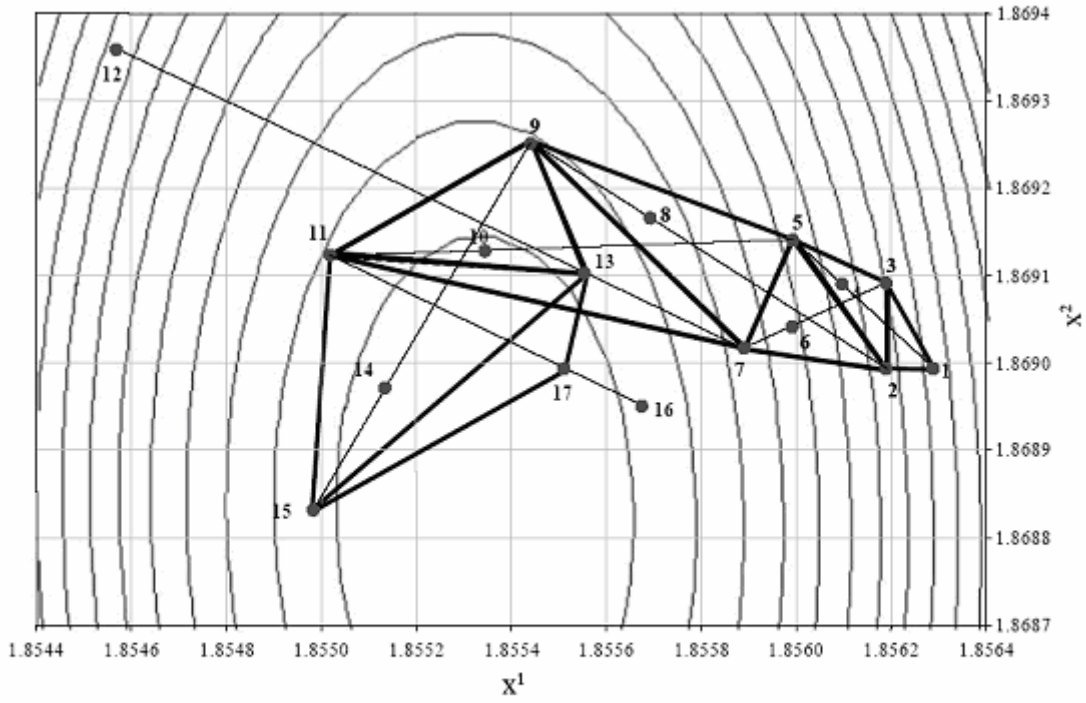


Fig. 7: Método simplex de Nelder y Mead.

El algoritmo se detiene cuando se alcanzan las condiciones de parada.

El primer criterio de parada es un test de la varianza de los valores de la función objetivo en los vértices del simplex.

$$\mathrm{var} f = \frac{1}{n}\left(\sum_{i=1}^{n+1}\left(f(x_i)\right)^2 - \frac{1}{n+1}\left(\sum_{i=1}^{n+1} f(x_i)\right)^2\right) < \epsilon^2.$$

Si la varianza de la función es menor que el cuadrado de un valor predefinido, entonces el algoritmo se detiene. Nelder y Mead eligieron este criterio de parada basándose en problemas estadísticos en los que hay que encontrar el mínimo de una superficie de suma de cuadrados. En estos problemas la curvatura cerca del mínimo proporciona información acerca de los parámetros desconocidos. Una curvatura suave indica un alto rango de varianza en la estimación, y por consiguiente, no hay porqué buscar el mínimo con gran exactitud. Sin embargo si la curvatura es considerable, entonces el rango de la varianza es bajo y la exactitud requerida para determinar los parámetros libres debe ser alta.

O'Neill modifica el criterio de parada añadiendo una nueva condición.

Comprueba si algún paso ortogonal, cada uno comenzando desde el mejor vértice del simplex actual, conduce a una mejora de la función objetivo. En caso afirmativo se reinicializa de nuevo el simplex desde ese nuevo vértice encontrado y se comienza de nuevo la búsqueda. En caso negativo la optimización concluye y la solución final viene dada por el mejor vértice encontrado antes de la búsqueda ortogonal.

La gran ventaja de este algoritmo es su rapidez de convergencia y su punto más débil es que sólo puede usarse para variables continuas.

2.2.4. Método Particle Swarm Optimization.

La gran ventaja de este algoritmo es que puede emplearse tanto para variables continuas como para variables discretas. En este aspecto, es muy útil para la definición de las variables que determinan el tipo de ventana. Su punto más débil es que necesita de muchas iteraciones para que se pueda garantizar una buena solución.

Se trata de un método heurístico que funciona de un modo similar a como lo hacen los algoritmos genéticos. Existe un grupo de individuos (soluciones factibles del problema de optimización entre las cuales no se encuentra necesariamente el óptimo) que forman una población (conjunto de soluciones factibles). Esta población va evolucionando con el tiempo y se van formando nuevas generaciones de individuos (nuevas soluciones), que se suponen que van mejorando con respecto a la generación anterior. De esta manera, al cabo de un número suficiente de generaciones, los individuos estarán muy próximos al óptimo. En la evolución de los individuos en cada generación, existen unos factores aleatorios que hacen que cada vez que se repita el algoritmo no necesariamente tiene que obtenerse la misma solución. En cualquier caso, para poblaciones con suficientes individuos, al cabo de un elevado número de generaciones prácticamente se puede garantizar que se alcanzará el óptimo.

2.3. Programa de simulación térmica de edificios de LIDER.

LIDER es la implementación informática de la opción general de verificación de la exigencia de Limitación de demanda energética (HE1), establecida en el Documento Básico de Habitabilidad y Energía del Código Técnico de la Edificación.

Ha sido desarrollado por AICIA - Grupo de Termotecnia para la Dirección General de Arquitectura y Política de Vivienda del Ministerio de la Vivienda y el Instituto para la Diversificación y Ahorro de la Energía (IDAE) del Ministerio de Industria, Comercio y Turismo.

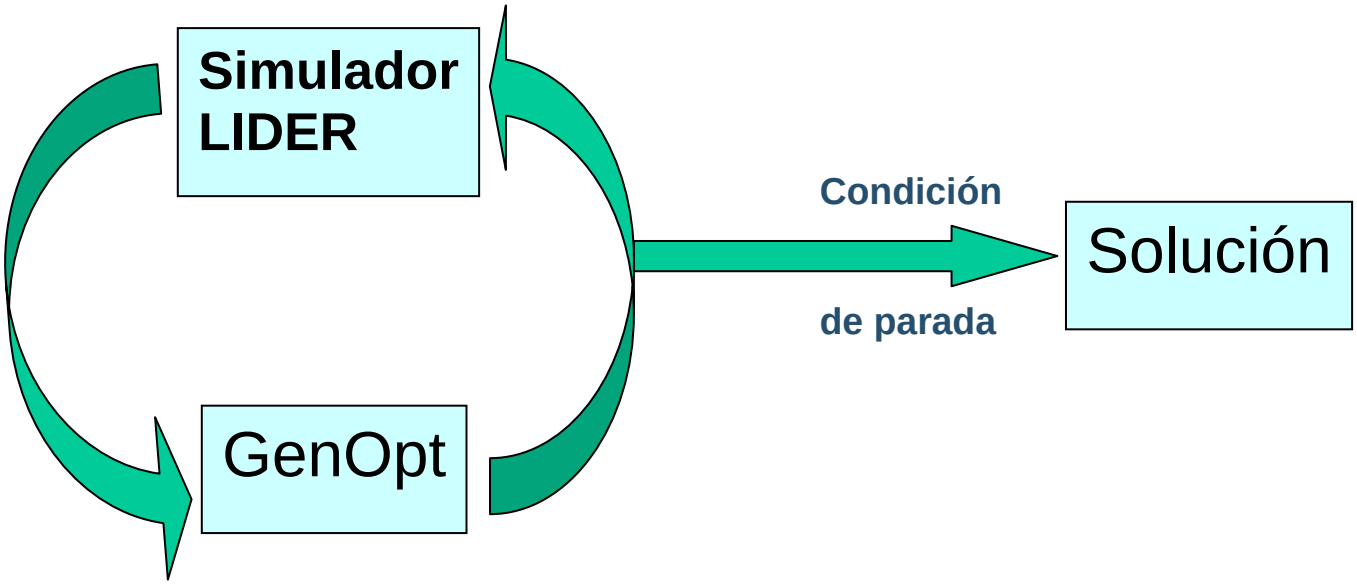
No lo emplearemos para la verificación de la exigencia Limitación de demanda energética (HE1), sino como simulador externo para realizar el cálculo térmico del edificio para evaluar la función objetivo. Para ello, recogeremos la información de los archivos de resultados (archivos de extensión .res) que genera el programa de cálculo de la demanda de LIDER.

Por otro lado, para definir las variables libres en cada problema de optimización que planteemos, debemos indicar adecuadamente algunas “señales” en el archivo de descripción del edificio en cuestión (archivos de extensión.CTE). En estas señales se irán cambiando automáticamente en cada iteración los valores de los parámetros antiguos por los valores nuevos generados por el algoritmo de GenOpt. Cuando se alcance el óptimo, el proceso se detiene y podemos disponer de los resultados correspondientes de las variables de más interés en nuestro problema.

2.4. Funcionamiento conjunto de GenOpt y el simulador de LIDER.

Nuestro objetivo es realizar la optimización del diseño de la epidermis edificatoria haciendo trabajar conjuntamente al programa de simulación térmica de edificios de LIDER con el programa GenOpt de optimización genérica.

El esquema general de funcionamiento del conjunto es:



Simulador de LIDER.

- Lee el archivo de texto con el nuevo conjunto de valores generado por GenOpt.
- Escribe un archivo de texto con el valor de la función objetivo.

GenOpt.

- Escribe un archivo de texto con el nuevo conjunto de valores a calcular por el simulador de LIDER, en base al algoritmo seleccionado.
- Lee de un archivo de texto el valor de la función objetivo, calculada a partir de los resultados de LIDER.

Para conectar los programas de simulación de LIDER y GenOpt, y así puedan funcionar conjuntamente, elaboraremos un código en C++ que deberá realizar las siguientes operaciones:

- Leer el archivo de las variables independientes (de extensión x.txt), generadas por GenOpt.

- Hacer las sustituciones con los valores de estas variables independientes en el archivo BDL de descripción del edificio en estudio.
- Llamar al programa de cálculo de la demanda del edificio.
- Leer el archivo de resultados del cálculo.
- Calcular la función objetivo.
- Escribir el archivo que deberá leer GenOpt (de extensión f.txt) con el valor de la función objetivo.
- Generar salidas auxiliares con datos de interés de demandas y costes calculados en cada iteración.

El código en C++ se adjunta en el anexo.

Una vez realizado el código en C++, analizaremos si es posible realizar la optimización del diseño de la envuelta de un edificio, comprobando que los resultados obtenidos son correctos.

Para ello, en primer lugar comenzaremos estudiando el caso de un edificio ficticio muy simple, en forma de cubo y únicamente con cuatro ventanas (una en cada fachada), con pocas variables libres. Con este edificio nuestra única pretensión es comprobar que el procedimiento seguido es correcto y que los resultados obtenidos son coherentes. De este modo, el tiempo de cálculo en cada iteración será pequeño (pocos segundos) y podremos realizar mayor cantidad de pruebas. Además este caso sencillo nos puede servir para afinar los parámetros del algoritmo seleccionado, logrando así encontrar el óptimo en el menor número posible de iteraciones. En este aspecto buscamos que cuando realicemos el análisis de otros edificios más complejos realicemos el menor número posible de iteraciones, ganando así tiempo.

En segundo lugar realizaremos algunas pruebas para el caso de edificios reales sencillos como viviendas adosadas y viviendas aisladas.

Finalmente estudiaremos un caso de un edificio más complejo, un bloque. El gran inconveniente en este tipo de edificios con mayor número de espacios es que el tiempo de ejecución para cada iteración es elevado, con lo que el análisis completo en un ordenador personal probablemente tarde muchas horas o incluso días.