Trabajo de evaluación

1. Objetivo

Realizar una implementación de un algoritmo multiobjetivo basado en agregación, con posibilidad de manejo de restricciones, y cuyas prestaciones sobre los problemas propuestos superen las capacidades del algoritmo utilizado en clases prácticas.

2. Descripción de algoritmo multiobjetivo basado en agregación

Consideremos sin pérdida de generalidad un problema multiobjetivo consistente en la minimización de *m* funciones objetivo (consideramos de momento que no hay restricciones):

$$\text{minimize } F(x) = (f_i(x), \cdots, f_m(x))$$

donde el espacio de búsqueda $x \in \Omega$ está definido mediante los límites inferior y superior de cada de las variables que lo componen:

$$x_{Li} \leqslant x_i \leqslant x_{Ui},\ i \in [1, p]$$

La filosofía de los algoritmos basados en agregación es descomponer el problema de optimización multi-objetivo en varios subproblemas de un único objetivo. Resulta obvio que la función objetivo de cada subproblema debe ser una cierta agregación de los objetivos del problema multi-objetivo. Hay distintas formulaciones posibles. Aquí utilizaremos la formulación de Tchebychef, definida de la siguiente manera:

$$g^{te}(x|\lambda, z^*) = \max_{1 \le i \le m} \left\{ \lambda_i \left| f_i(x) - z_i^* \right| \right\}$$

donde $\lambda = (\lambda_1, \cdots, \lambda_m)^T$ es un vector peso y $z^* = (z_1^*, \cdots, z_m^*)^T$ es un punto de referencia donde cada una de sus componentes está dado por:

$$z_i^* = \min \{ f_i(x) \quad x \in \Omega \}$$

3. Detalles de implementación de algoritmo basado en agregación

3.1 Parámetros de entrada establecidos por el usuario

- N
  Es el número de subproblemas, equivale al tamaño de la población que se utiliza en otros algoritmos.
- G
  Es el número de generaciones y actúa como mecanismo de detención: cuando hayan transcurrido G generaciones, el algoritmo se debe detener. Hay que tener en cuenta que en cada generación se evalúa un nuevo individuo correspondiente a cada subproblema, por lo que el número total de evaluaciones de objetivos y restricciones es N x G.
- Parámetros de control de los operadores evolutivos usados

En su implementación usará operadores evolutivos típicos, por operadores de mutación y cruce. Cada operador suele tener unos parámetros de configuración que habrá de utilizar en cada caso.

- T
  Es el tamaño de la vecindad de cada subproblema. Un valor recomendable es entre un 10% y un 30% del número de subproblemas.
- Espacio de búsqueda
  Está constituido por los límites inferior y superior de cada una de las *p* variables de búsqueda x_i del problema de que se trate:
  $x_{Li} \leqslant x_i \leqslant x_{Ui},\ i \in [1, p]$

3.2 Inicialización

Empezamos creando un conjunto de *N* vectores peso (uno por cada subproblema): $\lambda^1, \cdots, \lambda^N$ debiéndose cumplir que la suma de los componentes de cada vector es la unidad: $\sum_{i=1}^{m} \lambda_i^j = 1$ y que dichos vectores peso están distribuidos uniformemente (Nota: distribución uniforme no tiene nada que ver con funciones densidad de probabilidad, simplemente significa que los vectores están equiespaciados, donde la distancia se define de forma euclídea).

Calculamos la distancia euclídea entre cada pareja de vectores peso y para cada vector *i* $(i = 1, \cdots, N)$ identificamos el conjunto $B(i) = \{i_1, \cdots, i_T\}$ formados por los *T* vectores peso $\lambda^{i_1}, \cdots, \lambda^{i_T}$ más cercanos (desde el punto de vista de distancia euclídea) a λ^i (Nota: la vecindad incluye al propio individuo, dado que es el más cercano a sí mismo). Denominaremos a esto la vecindad de cada vector peso.

A continuación inicializamos una población de *N* individuos $x^1, \cdots, x^N$ y evaluamos sus prestaciones $F(x^i)$

Finalmente, como necesitaremos un punto de referencia y este es normalmente desconocido a priori, inicializamos un punto de referencia $z = (z_1, \cdots, z_m)^T$ donde z_i es el mejor valor encontrado de cada objetivo f_i. A medida que la ejecución del algoritmo progrese iremos actualizando este punto.

3.3 Acciones por iteración

Para cada iteración del algoritmo, se ejecuta la siguiente secuencia de acciones para cada subproblema *i*:

a) **Reproducción:** Seleccionamos aleatoriamente índices de su vecindad $B(i)$ y se aplican operadores evolutivos sobre los individuos asociados a esa vecindad para generar una nueva solución y. El número de índices a escoger y las operaciones a realizar sobre dichos individuos dependerán obviamente de los operadores evolutivos escogidos. Tenga en cuenta que los operadores de cruce y mutación actúan sobre el genotipo, por lo que no dependen de que el problema sea multi-objetivo o mono-objetivo. La filosofía detrás de esta operación es que

los subproblemas vecinos son relativamente parecidos por lo que sus soluciones posiblemente estén cerca de la del subproblema actual.

b) **Evaluación:** Evaluamos el individuo resultante $F(y)$

c) **Actualización del punto de referencia z:** Una vez obtenido y evaluado el nuevo individuo puede ser necesario actualizar el punto de referencia. Para todos los objetivos $j = 1, \cdots, m$, si $z_j > f_j(y)$ entonces $z_j = f_j(y)$.

d) **Actualización de vecinos:** A continuación se comprueba si la solución obtenida es mejor que la existente para cada uno de los vecinos. Si es así lo reemplaza. Matemáticamente: para cada $j \in B(i)$, si $g^{te}(y|\lambda^j, z) \le g^{te}(x^j|\lambda^j, z)$ entonces $x^j = y$. La idea aquí es que la solución obtenida puede no solamente ser buena para el subproblema actual sino también para subproblemas vecinos. Partiendo de la idea de que probar un individuo para un subproblema vecino es computacionalmente mucho más barato que la evaluación del individuo, se realiza dicha prueba.

3.4 Posibles operadores evolutivos

Puede utilizar y ensayar libremente (y se valorará positivamente) los operadores evolutivos que considere oportuno. No obstante, aquí se describe una propuesta. Consideramos en primer lugar los operadores evolutivos mutación y cruce del algoritmo de evolución diferencial. El operador de mutación requiere tres vectores:

$$v^{(i)}(G+1) = x^{(r1)}(G) + F \cdot (x^{(r2)}(G) - x^{(r3)}(G))$$

En el algoritmo de evolución diferencial estos tres vectores se escogían aleatoriamente. Aquí tiene sentido hacer lo mismo pero limitándonos a los subproblemas vecinos. A continuación se aplica el operador de cruce de evolución diferencial exactamente de la misma forma que el algoritmo de optimización de evolución diferencial. Valores razonables de los parámetros de mutación y cruce son: $F = 0.5$ y $CR = 0.5$.

A continuación aplicamos el operador de mutación gaussiana con probabilidad $PR = 1/p$ (recuerde que p es la dimensionalidad del espacio de búsqueda) a cada componente de la solución:

$$x_j' = x_j + N(0, \sigma_j)$$

donde $N(0, \sigma_j)$ es la función de densidad de probabilidad de Gauss con media 0 y desviación estándar σ_j. La desviación estándar viene dada por $\sigma_j = \frac{x_{Uj} - x_{Lj}}{SIG}$, donde $SIG = 20$.

A la hora de implementar este último operador puede encontrar la dificultad de que el lenguaje o plataforma elegida no disponga de generadores de números aleatorios siguiendo una función densidad de probabilidad gaussiana. Para solucionar este problema se puede recurrir a una técnica de generación matemática a partir de distribuciones uniformes. Supongamos que *A* y *B* son dos distribuciones uniformes en el rango [0,1]. Se puede construir una distribución gaussiana *U* con media 0 y desviación estándar 1 sin más que realizar la siguiente operación:

$$U = \sqrt{-2 \cdot \ln(A)} \cdot \cos(2\pi B)$$

Nota importante: Recuerde que en la sección 3.1 se ha definido el espacio de búsqueda, lo que significa que las variables no pueden exceder dicho espacio. Como consecuencia de la aplicación de operadores evolutivos, alguna variable puede exceder dichos límites. En ese caso, hay que actuar para confinar las variables dentro del espacio. De las distintas opciones posibles, aquí fijaremos el valor de la variable al valor límite.

3.5 Extensiones necesarias

El algoritmo basado en agregación que se ha definido hasta ahora no tiene restricciones. Para ser capaz de abordar aquellos problemas que tienen restricciones es necesario incorporar un mecanismo de manipulación de restricciones. Los dos mecanismos principales que se utilizan con algoritmos de optimización mono-objetivo pueden adaptarse fácilmente aquí:

1) Mecanismo basado en selección o separación de objetivos y restricciones. Cuando se comparan dos soluciones pueden darse tres situaciones:
   a) Las dos soluciones incumplen restricciones: se selecciona la que incumpla menos.
   b) Un solución satisface las restricciones y otra no: se selecciona la que cumple.
   c) Las dos soluciones cumplen las restricciones: se utiliza para la selección los mismos criterios que si se tratara de un problema sin restricciones.

   En este caso, es posible que la actualización de soluciones a los vecinos distintos de uno mismo no sea una buena idea. Puede ensayarlo.
2) Mecanismo basado en funciones penalti. De forma análoga a algoritmos de optimización mono-objetivo, se suma a las funciones objetivo una penalización del incumplimiento de las restricciones. Una vez que se aplica la transformación, se puede manejar como si no hubiera restricciones. No obstante, en el análisis de resultados debe examinar si se cumplen las restricciones o no.

3.6 Posibles extensiones

En algunas ocasiones se utiliza un archivo externo de soluciones. El archivo externo se inicializa tras los pasos de inicialización indicados anteriormente almacenando las soluciones no dominadas. En cada iteración, con cada generación de una nueva solución *y* se compara la solución obtenida con las existentes en el archivo. Si domina a alguna existente, se elimina. Si ninguna de las existentes domina a *y*, se añade al archivo. Tenga en cuenta que el tamaño del archivo puede crecer considerablemente por lo que es posible que tenga que incorporar criterios de control.

4. Definición de problemas de la competición

La implementación que realice deberá aplicarla a los siguientes problemas:

1) ZDT3 (30 dimensiones)
2) CF6 (4 y 16 dimensiones)

En todos los casos considerará que dispone de un presupuesto de:

a) 10000 evaluaciones
b) 4000 evaluaciones

Se entiende por evaluación cada vez que evalúe los objetivos y restricciones (si las hubiere) de los problemas ZDT3 y CF6, es decir, el número total de evaluaciones viene dado por el producto del tamaño de la población y el número de generaciones o iteraciones del algoritmo.
Deberá comparar las prestaciones del algoritmo implementado con las del algoritmo NSGAII con las mismas condiciones (comparación justa). La comparación debe incluir el uso de métricas. Recuerde que se trata de algoritmos estocásticos por lo que es imprescindible realizar estadísticas de múltiples ejecuciones.
Para facilitarle las comparaciones, en Enseñanza Virtual tiene el resultado de la aplicación de NSGAII a los problemas anteriores. Encontrará una carpeta con el nombre de cada uno de los problemas anteriores. Dentro de cada una de ellas tiene dos paquetes de archivos, uno para cada presupuesto de evaluaciones: 10000 o 4000. Cuando descomprima cada uno de ellos encontrará varios directorios de nombre PxxGyy, donde "xx" se refiere al número de individuos usados en NSGAII y "yy" al número de generaciones. Dentro de cada directorio tiene los resultados de 10 ejecuciones, cada una con una semilla (seed) diferente. Para cada una de las ejecuciones hay un fichero "allpopm…" y un fichero "final_pop…". El fichero "allpopm…" puede usarse directamente como fichero de entrada para el software de métricas de comparación. El fichero "final_pop…" contiene los datos de la última generación y puede usarse para representar el frente obtenido.

5. Entrega de resultados

Los resultados se entregarán a través de la plataforma de Enseñanza Virtual con antelación a la fecha límite indicada. El material a entregar incluirá al menos:

1) Código realizado en el lenguaje o plataforma elegido.
2) Descripción de implementación mostrando los criterios y decisiones tomadas. Este es un documento técnico del algoritmo implementado, no una descripción de funciones en el lenguaje elegido. El documento debe ser concreto y detallado, sobre la implementación específica que el alumno ha realizado. Debe indicar cómo lo ha hecho y por qué. El documento ha de ser autocontenido, es decir, examinar secciones del código no debe ser necesario para comprender el documento en su totalidad. En ningún caso este documento se debe construir con capturas del código realizado. Es positivo documentar todas las opciones elegidas, incluso si no han sido exitosas. La información sobre lo que no funciona también es un mensaje útil.
3) Documento con resultados comparativos sobre los problemas indicados en la Sección 4. Debe representar los frentes (incluyendo superposición de los obtenidos con NSGAII y el algoritmo implementado para poder apreciar nítidamente las diferencias; obviamente solo tiene sentido representar los frentes finales, no el conjunto de soluciones que se han ido generando en iteraciones intermedias) así como el uso de métricas (spacing, hipervolumen y coverage set) y estadísticas de todos ellos. Deberá quedar perfectamente claro cuáles son los frentes que se están comparando y las condiciones: por ejemplo, no tiene sentido proporcionar un valor de hipervolumen si no se dice cuál es el punto de referencia y cómo se ha obtenido. El documento debe incluir el análisis y discusión crítica de los resultados, nunca se debe limitar a una sucesión de gráficas y datos. Los datos numéricos sobre frentes y métricas obtenidas deben proporcionarse en ficheros independientes convenientemente referenciados, pero no deben ser necesarios para la comprensión completa del documento.

Tras la entrega de resultados se podrá concertar una entrevista.