UNIVERSIDAD DE MURCIA

Resolución de un nuevo modelo biobjetivo para la localización de un centro semi-repulsivo mediante algoritmos evolutivos

Laura Antón Sánchez

Trabajo Fin de Máster

Máster en Matemática Avanzada y Profesional

Universidad de Murcia

Curso 2011/12

D. José Fernández Hernández, Profesor Titular del Departamento de Estadística e Investigación Operativa de la Universidad de Murcia

AUTORIZA:

La presentación del Trabajo Fin de Máster titulado *Resolución de un nuevo modelo biobjetivo para la localización de un centro semi-repulsivo mediante algoritmos evolutivos*, realizado por Dª. Laura Antón Sánchez, bajo su inmediata dirección y supervisión, dentro del Máster en Matemática Avanzada y Profesional de la Universidad de Murcia.

En Murcia, a 13 de Julio de 2012

Fdo: José Fernández Hernández

Índice General

Prefacio

La preocupación por los problemas medioambientales ha estado y continúa estando en aumento en los últimos años. Este trabajo se centra en el estudio de un modelo de localización de una instalación contaminante, a la que se ha puesto el calificativo de *semi-repulsiva* por resultar atractiva para determinados individuos pero ser no deseada por otros.

El centro semi-repulsivo debe estar situado dentro de una región geográfica determinada y se persigue, simultáneamente, minimizar los costes de transporte a las instalaciones, así como minimizar la oposición global de los habitantes de esa región a las mismas. Por ello, el modelo se ha traducido en un problema de optimización biobjetivo, donde las preocupaciones medioambientales se han tenido en cuenta a través de la definición de áreas protegidas donde no se admite la ubicación de las instalaciones y regiones prohibidas alrededor de cada ciudad para evitar la localización muy cerca de ellas.

El objetivo de este trabajo es el análisis de este modelo mediante su resolución haciendo uso de técnicas metaheurísticas multiobjetivo. Por ello, el primer capítulo está dedicado a la introducción de conceptos básicos de optimización multiobjetivo y de heurísitcas basadas en poblaciones, en concreto, de algoritmos genéticos y búsqueda dispersa. En el segundo capítulo se desarrollan los algoritmos genéticos multiobjetivo con los que se resolverán los problemas de localización de centros semi-repulsivos, y que son los algoritmos NSGA-II [5] (Nondominated Sorting Genetic Algorithm II), SPEA2 [27] (Strength Pareto Evolutionary Algorithm 2) y AbYSS [17] (Archived-Based hYbrid Scatter Search). En el capítulo tercero se presenta formalmente el modelo de localización de centros semi-repulsivos y en el cuarto capítulo se presentan estudios computacionales, describiéndose cómo se han generado y ejecutado una amplia variedad de estos problemas de localización, así como el análisis de los resultados obtenidos. Por último, en el capítulo cinco se dan algunas conclusiones finales.

1

Preliminares

1.1 Optimización multiobjetivo

Existen muchas situaciones reales en las que las personas no se rigen por un único criterio a la hora de tomar una decisión y, obviamente, la mejora de algunos de estos criterios puede ir en detrimento de otros. El primer problema que surge en este planteamiento es el hecho de que, en general, no existirá una solución óptima, en el sentido de ser la que maximice o minimice todos los criterios considerados. Las técnicas que estudian la gestión de este tipo de problemas se denominan técnicas de programación multiobjetivo o multicriterio, haciendo referencia a la existencia de más de un criterio a tener en cuenta en la selección de alternativas.

En términos generales, la optimización multiobjetivo no se limita a la búsqueda de una solución única para un determinado problema de optimización, sino más bien a encontrar el conjunto de las llamadas soluciones *eficientes*. Cuando dichas soluciones se dibujan en el espacio objetivo son conocidas colectivamente como *frente de Pareto*. El objetivo principal de la optimización multiobjetivo es la obtención del frente de Pareto, lo que significa que los optimizadores multiobjetivo necesitan explorar grandes porciones del espacio de búsqueda ya que buscan no un único óptimo, sino el conjunto de Pareto óptimo. Además, muchos problemas de optimización multiobjetivo reales suelen necesitar métodos computacionalmente costosos para el cálculo de sus funciones objetivo y restricciones.

Considérese un problema de optimización multiobjetivo (POM) de la forma

$$\begin{array}{ll} \text{minimizar} & \{f_1(x), f_2(x), ..., f_k(x)\} \\ \text{sujeto a} & x \in S \end{array} \tag{1.1}$$

donde hay k (≥ 2) funciones objetivo $f_i : \Re^n \to \Re$. $f(x) = (f_1(x), f_2(x), ..., f_k(x))^T$ es el vector de funciones objetivo. El vector $x = (x_1, x_2, ...x_n)^T$ de variables de decisión debe pertenecer al conjunto (no vacío) factible S, un subconjunto de $\Re^n$.

Con *minimizar* se hace referencia a que se quiere minimizar todas las funciones objetivo simultáneamente. Si no hay conflicto entre las funciones objetivo, puede encontrarse una solución donde todas las funciones objetivo alcancen su óptimo. En este caso, no se requiere ningún método especial. Para evitar estos casos triviales, asumiremos que no existe una solución única que es óptima respecto de todas las funciones objetivo. Esto significa que las funciones objetivo están, al menos en parte, en conflicto.

La imagen de la región factible (subconjunto del espacio objetivo $\Re^k$) se denota por Z ($= f(S)$). Los elementos de Z se llaman vectores objetivo y se denotan por $f(x)$ o $z = (z_1, z_2, ..., z_k)^T$, donde $z_i = f_i(x)$, $i = 1, ..., k$, son los valores objetivo.

Por claridad y simplicidad, supondremos que todas las funciones objetivo tienen que ser minimizadas. Si una función objetivo f_i tuviera que ser maximizada, es equivalente a minimizar la función $-f_i$.

En problemas de optimización con un solo objetivo, el interés se centra en el espacio de las variables de decisión. En cambio, en optimización multiobjetivo a menudo se está más interesado en el espacio objetivo puesto que, normalmente, es de una dimensión menor. Debido a que las funciones objetivo en un POM suelen estar en conflicto, no es posible encontrar una solución única que sea óptima para todas simultáneamente. No obstante, el estudio de algunos de los vectores objetivo resulta de interés. Estos vectores son aquellos en los que ninguna de las componentes puede ser mejorada sin deteriorar, al menos, una de los otras. Edgeworth presentó esta definición en 1881 [8]. Sin embargo, la definición se conoce comúnmente como óptimo de Pareto después de que el economista y sociólogo Vilfredo Pareto desarrollara aún más este concepto en 1896 [18, 19]. Una definición más formal sería la siguiente:

Definición 1.1 *Un vector de decisión $x^* \in S$ se dice* eficiente *si no existe otro vector de decisión $x \in S$ tal que $f_i(x) \leq f_i(x^*)$ para todo $i = 1, ..., k$ y $f_j(x) < f_j(x^*)$ para al menos un índice j.*

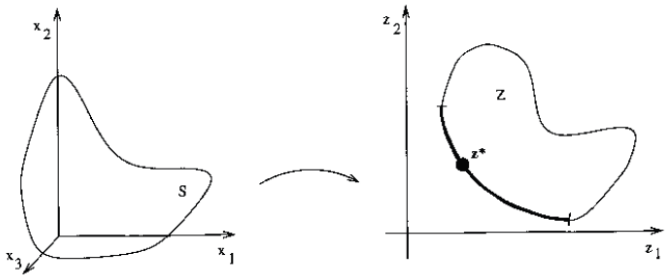

Figura 1.1: Conjuntos S y Z y frente de Pareto

Un vector objetivo $z^* \in Z$ se dice que es un óptimo de Pareto *si no existe otro vector objetivo $z \in Z$ tal que $z_i \leq z_i^*$ para todo $i = 1, .., k$ y $z_j < z_j^*$ para al menos un índice j; o equivalentemente, z^* es un óptimo de Pareto si su vector de decisión correspondiente es eficiente.*

En la figura 1.1 se muestra una región factible $S \subset \Re^3$ y su imagen, una región objetivo factible $Z \subset \Re^2$. La línea más gruesa contiene todos los vectores objetivo óptimos de Pareto. El vector z^* es un ejemplo de ellos.

La definición 1.1 dice que x^* es eficiente si no existe otro vector factible x que mejore algún criterio sin hacer que empeore al menos otro de ellos, es decir, que lo domine, por lo que se conoce a x^* como *solución no dominada*. Además, se conoce como frente de Pareto a las imágenes de los puntos eficientes. Aproximar dicho frente es precisamente el objetivo de la optimización multiobjetivo.

1.2 Heurísticas basadas en poblaciones

1.2.1 Algoritmos genéticos

Los Algoritmos Genéticos (AGs) son métodos de búsqueda/optimización estocásticos (no determinísticos) que utilizan las teorías de la evolución y de la selección natural para resolver un problema dentro de un espacio de soluciones complejas. Hoy día es común utilizar el término de *computación evolutiva* o *algoritmos evolutivos* (AEs), con el fin de abarcar los desarrollos de los últimos 10 años.

El término *algoritmo genético*, fue utilizado por primera vez por John Holland [13], cuyo libro *Adaptation in Natural and Aritificial Systems* en 1975 jugó un papel

decisivo en la creación de lo que hoy es un próspero campo de investigación y aplicación que va mucho más allá del AG original. Holland afirmaba que los procesos de adaptación son, básicamente, procesos de optimización, pero que es difícil someterlos a un estudio unificado porque las estructuras que se modifican son complejas y su desempeño es incierto. El enfoque de su libro era la creación de un marco matemático que hiciera posible extraer y generalizar los factores críticos de los procesos biológicos. Dos de las generalizaciones más importantes eran: la generalización de un *esquema* para la interacción de un conjunto de genes coadaptados, y la generalización de los operadores genéticos como son el cruce y la mutación. El concepto de *esquema* permite diseccionar y analizar complejos problemas "no lineales", mientras que la generalización de los operadores genéticos permite extender el análisis a estudios de aprendizaje, planificación óptima, etc.

La idea común de los trabajos de Holland y de otros científicos involucrados en desarrollos similares fue el uso de la mutación y la selección, conceptos fundamentales en la teoría de la evolución neodarwinista. A pesar de que se obtuvieron algunos resultados prometedores, la computación evolutiva no despegó realmente hasta la década de los 80s, siendo uno de los motivos que las técnicas que se necesitaban requerían una gran potencia de cálculo. En 1989, el libro *Genetic Algorithms in Search, Optimization, and Machine Learning* de David Goldberg [12], un estudiante de Holland, puso las bases para el establecimiento de un desarrollo sostenido de la teoría de AGs y su aplicación en problemas prácticos, que sigue creciendo rápidamente.

Los Algoritmos Genéticos trabajan sobre una población de individuos, donde cada uno de ellos representa una posible solución al problema que se desea resolver. Todo individuo tiene asociado un ajuste de acuerdo a la bondad con respecto al problema de la solución que representa, conocido como *fitness*. El funcionamiento genérico de un AG puede observarse en el algoritmo 1.

Los diversos puntos y etapas a tener en cuenta en la aplicación de un AG se describen a continuación [11]:

Codificación

Cualquier solución potencial a un problema puede ser presentada dando valores a una serie de parámetros. El conjunto de todos los parámetros (genes en la terminología de AGs) se codifica en una cadena de valores denominada cromosoma. El conjunto de los parámetros representado por un cromosoma particular recibe el nombre de genotipo. El genotipo contiene la información necesaria para la construc-

Algoritmo 1 - Esquema básico de un algoritmo genético

```
1  Elegir una población inicial de cromosomas
2  while (no se satisfaga la condición de parada) do
3      repeat
4          if (se satisface la condición de cruce) then
5              Seleccionar los cromosomas padre. Cruzar
6          end if
7          if (se satisface la condición de mutación) then
8              Elegir los puntos de mutación. Mutar
9          end if
10         Evaluar a los hijos
11     until suficientes hijos creados
12     Seleccionar una nueva población
13 end while
```

ción del organismo, es decir, la solución real al problema, denominada fenotipo. En términos biológicos, la información genética contenida en el ADN de un individuo sería el genotipo, mientras que la expresión de ese ADN (el propio individuo) sería el fenotipo.

Desde los primeros trabajos de Holland la codificación suele hacerse mediante valores binarios. Se asigna un determinado número de bits a cada parámetro y se realiza una discretización de la variable representada por cada gen (evidentemente no todos los parámetros tienen que estar codificados con el mismo número de bits). Cada uno de los bits pertenecientes a un gen suele recibir el nombre de alelo. También pueden existir representaciones que codifiquen directamente cada parámetro con un valor entero, real o en punto flotante, permitiendo el desarrollo de operadores genéticos más específicos.

Población inicial

Dos importantes cuestiones a considerar son, en primer lugar, el tamaño de la población y, en segundo lugar, el método por el cual los individuos son seleccionados. El tamaño de la población debería elegirse de modo que hubiera un equilibrio entre eficiencia y eficacia, es decir, una población demasiado pequeña no aportaría margen suficiente para explorar el espacio de búsqueda eficazmente, mientras que una demasiado grande perjudicaría la eficiencia del método al no poder obtener una solución en un período razonable de tiempo.

Según [20] el tamaño mínimo de la población para que tenga lugar una búsqueda significativa debería ser aquel con el que, al menos, cada punto del espacio de

búsqueda pudiera ser accesible desde la población inicial mediante únicamente la operación de cruce. Este requisito sólo puede ser satisfecho si en el conjunto completo de cadenas de la población existe en cada gen por lo menos una ocurrencia de cada alelo, es decir, de cada variante del gen. En el supuesto de que la población inicial sea generada mediante una muestra aleatoria con reemplazamiento, se puede encontrar la probabilidad de que al menos un alelo esté presente en cada gen. Los resultados de [20] sugieren que un tamaño de la población de $O(\log l)$, con l la longitud de las cadenas representando a los individuos, sería suficiente para cubrir el espacio de búsqueda.

Referente a cómo elegir la población, casi siempre se supone que la inicialización debe ser al azar. Evidentemente los puntos elegidos al azar no necesariamente cubren el espacio de búsqueda de un modo uniforme y podría ser beneficioso, en términos de cobertura, si usáramos métodos estadísticos más sofisticados, especialmente para los alfabetos no binarios.

Otro punto a tener en cuenta es la posibilidad de utilizar una *semilla* en la población inicial con soluciones buenas ya conocidas. La inclusión de una solución de alta calidad, obtenida mediante, por ejemplo, otra técnica heurística, puede ayudar a un AG a encontrar mejores soluciones de una forma más rápida de lo que pudiera hacerlo con un comienzo aleatorio. Sin embargo, existe también la posibilidad de inducir convergencia prematura.

Condición de parada

A diferencia de métodos de búsqueda más simples que terminan cuando alcanzan un óptimo local, los AGs son métodos estocásticos de búsqueda que podrían, en principio, seguir iterando para siempre. Por ello, en la práctica, se necesita un criterio de parada, comúnmente establecer un número máximo de evaluaciones o un límite de tiempo, pero también se podría realizar un seguimiento de la diversidad de la población y detener el procedimiento cuando ésta cae por debajo de un umbral preestablecido.

Condición de cruce y mutación

En un algoritmo genético, a la hora de decidir si se aplican las operaciones de cruce y mutación para generar nuevos individuos, podría seguirse una estrategia de cruce-Y-mutación o utilizar una de cruce-O-mutación. La primera de ellas intenta, en primer lugar, llevar a cabo el cruce y, a continuación, intenta la mutación en los hijos generados (ya sea sólo en uno de ellos o en ambos). Con esta estrategia en algunos casos puede ocurrir que no se realice ninguna de las dos operaciones,

con lo que los descendientes serían simples clones de los padres. En cambio, en las estrategias de cruce-O-mutación siempre se hace algo, ya sea cruce o mutación, pero no ambos. En este caso también podría darse el caso de obtener clones de los padres al cruzar si éstos son demasiado parecidos.

El mecanismo para la implementación de estas decisiones es habitualmente una regla aleatoria, mediante la cual la operación se lleva a cabo si se excede un umbral determinado. En el caso de la mutación, podemos elegir el número de mutaciones por cadena o por bit (mutación bit a bit). En el caso de la estrategia de cruce-O-mutación, existe la posibilidad adicional de modificar la proporción de cruces y mutaciones a medida que avanza la búsqueda, por ejemplo, una alta proporción de cruces al inicio y aumento de la mutación a medida que la población converge. Estas proporciones podrían incluso adaptarse en el transcurso de la búsqueda, dependiendo de los cromosomas de alta calidad encontrados.

Selección

Los algoritmos de selección serán los encargados de escoger qué individuos van a disponer de oportunidades de reproducirse y cuáles no, otorgando un mayor número de oportunidades de reproducción a los individuos más aptos. Por tanto, la selección de un individuo estará relacionada con su valor de ajuste o *fitness*, aunque debemos tener en cuenta que no se debe eliminar por completo las opciones de reproducción de los individuos menos aptos, pues en pocas generaciones la población podría volverse homogénea. Una opción podría ser seleccionar el primero de los individuos participantes en el cruce mediante alguno de los métodos comentados a continuación y el segundo de manera aleatoria.

La ***selección por ruleta*** es posiblemente el método más utilizado. A cada uno de los individuos de la población se le asigna una parte de una ruleta proporcional a su *fitness*, de tal forma que la suma de todos los porcentajes sea la unidad. Los mejores individuos recibirán una porción de la ruleta mayor que la recibida por los peores. Generalmente la población está ordenada en base al ajuste por lo que las porciones más grandes se encuentran al inicio de la ruleta. Para seleccionar un individuo basta con generar un número aleatorio del intervalo $[0, 1]$ y devolver el individuo situado en esa posición de la ruleta. Esta posición se suele obtener recorriendo los individuos de la población y acumulando sus proporciones de ruleta hasta que la suma exceda el valor obtenido. Es un método muy sencillo pero ineficiente a medida que aumenta el tamaño de la población, y presenta además el inconveniente de que el peor individuo puede ser seleccionado más de una vez.

En la ***selección por torneo*** la idea principal consiste en realizar la selección en base a comparaciones directas entre individuos. Existen dos versiones de selección mediante torneo, la determinística y la probabilística. En la versión determinística se selecciona al azar un número p de individuos (generalmente $p = 2$) y de entre los individuos seleccionados se selecciona el más apto para pasarlo a la siguiente generación. La versión probabilística únicamente se diferencia en el paso de selección del ganador del torneo. En vez de escoger siempre el mejor, se genera un número aleatorio en el intervalo $[0, 1]$; si es mayor que un parámetro p (fijo para todo el proceso evolutivo) se escoge el individuo más apto y en caso contrario el menos apto. Variando el número de individuos que participan en cada torneo se puede modificar la presión de selección. Cuando participan muchos individuos en cada torneo, la presión de selección es elevada y los peores individuos apenas tienen oportunidades de reproducción. Un caso particular es el *elitismo global*, torneo en el que participan todos los individuos de la población, con lo cual la selección se vuelve totalmente determinística.

Cruce

El cruce consiste simplemente en sustituir algunos de los genes de uno de los padres por los genes correspondientes del otro. Supongamos que tenemos 2 cadenas **a** y **b**, cada una compuesta de 6 variables (de cualquier alfabeto) que representan dos posibles soluciones a un problema:

$$(a_1, a_2, a_3, a_4, a_5, a_6) \text{ y } (b_1, b_2, b_3, b_4, b_5, b_6)$$

El cruce por un punto, ***SPX*** por sus siglas en inglés (Single Point Crossover), es la más sencilla de las técnicas de cruce. Seleccionados los dos individuos, se cortan sus cromosomas por un punto elegido aleatoriamente, generando dos segmentos diferenciados en cada uno de ellos, la cabeza y la cola, y se intercambian las colas entre los individuos para generar los nuevos descendientes. De esta forma, ambos descendientes heredan información genética de los padres.

El cruce por dos puntos es una generalización del cruce por un punto. En vez de cortar por un único punto los cromosomas de los padres como en el caso anterior, se realizan dos cortes (habría que tener en cuenta que ninguno de estos puntos de corte coincida con el extremo de los cromosomas para garantizar que se originan tres segmentos). Para generar la descendencia se escoge el segmento central de uno de los padres y los segmentos laterales del otro padre. Generalmente se suele referir a este tipo de cruce con las siglas ***DPX*** (Double Point Crossover). Generalizando se pueden añadir más puntos de cruce dando lugar a algoritmos de cruce multipunto.

Otra alternativa es el cruce uniforme, donde cada gen de la descendencia tiene las mismas probabilidades de pertenecer a uno u otro padre. La técnica implica la generación de una máscara de cruce con valores binarios. Si en una de las posiciones de la máscara hay un 1, el gen situado en esa posición en uno de los descendientes se copia del primer padre, si por el contrario hay un 0 el gen se copia del segundo padre. Para producir el segundo descendiente se intercambian los papeles de los padres, o bien se intercambia la interpretación de los unos y los ceros de la máscara de cruce. Se suele referir a este tipo de cruce con las siglas ***UPX*** (Uniform Point Crossover).

Para algoritmos genéticos con codificación real, un operador de cruce comúnmente utilizado es el ***SBX*** (Simulated Binary Crossover), propuesto por Deb [3] y que intenta emular el efecto del cruce por un punto usado en representación binaria. Este cruce genera dos hijos β^{s_1} y β^{s_2} a partir de los padres β^{f_1} y β^{f_2}:

$$\begin{array}{l} \beta^{s_1} = \frac{1}{2}[(1+B_k)\beta^{f_1} + (1-B_k)\beta^{f_2}] \\ \beta^{s_2} = \frac{1}{2}[(1-B_k)\beta^{f_1} + (1+B_k)\beta^{f_2}] \end{array} \tag{1.2}$$

donde $B_k \geq 0$ es un valor generado según la función de densidad

$$p(B) = \begin{cases} \frac{1}{2}(\eta+1)B^{\eta}, & \text{si } 0 \leq B \leq 1 \\ \frac{1}{2}(\eta+1)\frac{1}{B^{\eta+2}}, & \text{si } B > 1 \end{cases} \tag{1.3}$$

Esta distribución puede ser obtenida fácilmente a partir de una distribución uniforme $u(0,1)$ mediante la transformación

$$B(u) = \begin{cases} (2u)^{\frac{1}{\eta+1}}, & \text{si } u \leq \frac{1}{2} \\ (\frac{1}{2(1-u)})^{\frac{1}{\eta+1}}, & \text{si } u > \frac{1}{2} \end{cases} \tag{1.4}$$

El parámetro η establece la amplitud de la distribución con la que se generan los descendientes.

Además, hay muchos otros aspectos a tener en cuenta en la aplicación del cruce, tales como la frecuencia con que se aplica (algunas estrategias siempre lo hacen y otras utilizan un enfoque estocástico realizando el cruce con una probabilidad $\chi < 1$), decidir si se generan dos descendientes o uno, si elegimos sólo uno de los dos cómo realizar esta elección, etc.

Mutación

Si nos encontramos en el caso de la estrategia de cruce-O-mutación, debemos decidir en primer lugar si la mutación se lleva a cabo. Suponiendo que así es, el concepto de mutación es simple y puede ser representado como una cadena de bits. Si generamos una máscara como por ejemplo:

0 1 0 0 1 0

utilizando una distribución de Bernoulli en cada gen (con un valor pequeño de p), el ejemplo indicaría que a los genes 2º y 5º se les asigna un nuevo valor.

Hay diferentes formas de implementar esta simple idea, pudiendo reflejar una diferencia sustancial en el desempeño de un AG. Una primera idea podría ser generar un número aleatorio para cada gen de la cadena y compararlo con un umbral establecido μ, pero esto es costoso en términos de cómputo si las cadenas son largas y la población es grande. Una alternativa más eficaz es establecer una variable aleatoria con distribución de Poisson con parámetro λ, donde λ es el número medio de mutaciones por cromosoma. Un valor común para λ es 1, es decir, si l es la longitud de la cadena, $\mu = 1/l$. Tras haber decidido que hay m mutaciones, generamos m números al azar (sin reemplazamiento) uniformemente distribuidos entre 1 y l para especificar los lugares donde se va a realizar la mutación.

En el caso de cadenas binarias, la mutación significa asignar a los bits que corresponda el valor contrario al que tengan. De manera más general, cuando hay varios alelos posibles para cada gen, si decidimos cambiar un alelo particular, debemos proporcionar algún medio para decidir cuál debe ser su nuevo valor. Esto podría realizarse mediante una elección aleatoria, pero si (como en algunos casos) hay cierta relación ordinal entre los valores de los alelos, puede ser más sensato restringir las opciones en la elección de los alelos a aquellos que están cerca del valor actual o, al menos, sesgar la distribución de probabilidad a su favor.

Este es el caso de la mutación polinomial en variables continuas [4], donde el valor actual de la variable se cambia a un valor cercano utilizando una distribución de probabilidad polinomial con media el valor actual y varianza una función del índice de distribución η. Para realizar la mutación, se define una factor de perturbación δ como sigue:

$$\delta = \frac{c - p}{\Delta_{max}} \tag{1.5}$$

donde Δ_{max} es una cantidad fija que representa la máxima perturbación admisible

en el valor padre p y c es el valor mutado. El valor de mutación se calcula con una distribución de probabilidad que depende del factor de perturbación δ:

$$P(\delta) = \frac{1}{2}(\eta + 1)(1 - |\delta|)^{\eta} \tag{1.6}$$

La distribución de probabilidad anterior es válida en el rango $\delta \in (-1, 1)$. Para crear un valor mutado, se genera un número aleatorio u en el intervalo $(0, 1)$. A continuación, la siguiente ecuación puede ser utilizada para calcular el factor de perturbación $\bar{\delta}$ correspondiente a u utilizando la distribución de probabilidad anterior:

$$\bar{\delta} = \begin{cases} (2u)^{\frac{1}{\eta+1}} - 1, & \text{si } u < \frac{1}{2} \\ 1 - [2(1-u)]^{\frac{1}{\eta+1}}, & \text{si } u \geq \frac{1}{2} \end{cases} \tag{1.7}$$

Posteriormente, el valor mutado se calcula como:

$$c = p + \bar{\delta}\Delta_{max} \tag{1.8}$$

Evaluación

Para el correcto funcionamiento de un AG debe existir un método que indique si los individuos de la población representan o no buenas soluciones al problema planteado. De esto se encarga la función de evaluación que establece una medida numérica de la bondad de una solución. La aproximación más común consiste en crear explícitamente una medida de ajuste o *fitness* para cada individuo de la población. A cada uno de los individuos se les asigna un valor de ajuste escalar por medio de un procedimiento de evaluación bien definido, siendo este procedimiento de evaluación específico del dominio del problema en el que se aplica el AG.

Nueva población

El AG original de Holland asumía un enfoque generacional: selección, cruce y mutación se aplicaban a una población de N cromosomas hasta que un nuevo conjunto de N individuos eran generados y este conjunto pasaba a ser la nueva población. Desde el punto de vista de la optimización, podríamos pensar que el esfuerzo realizado para obtener una buena solución podría ser desaprovechado ya que corremos el riesgo de desecharla evitando su participación en la reproducción. Por esta razón, De Jong [2] introdujo los conceptos de *elitismo* y de *superposición en la población*. Sus ideas son sencillas. Una estrategia elitista asegura la supervivencia del mejor individuo hasta el momento, reemplazando por tanto sólo los restantes $(N-1)$

miembros de la población con nuevas cadenas. La superposición en la población lleva este concepto un paso más allá, mediante la sustitución de únicamente una fracción de la población en cada generación. Por último, tendríamos las denominadas estrategias de estado estacionario o incremental, en el que sólo un nuevo cromosoma (o a veces un par) se generan en cada etapa.

En el caso de una reproducción incremental también es necesario seleccionar los miembros de la población que van a eliminarse. Algunos AGs asumen que los padres son sustituidos por sus hijos. Muchas implementaciones, eliminan las peores miembros de la población, aunque debe tenerse en cuenta que esto ejerce una presión selectiva muy fuerte en la búsqueda, pudiendo ser necesaria una población bastante grande y altas tasas de mutación para evitar una rápida pérdida de la diversidad. Un método más suave es eliminar un determinado porcentaje de los peores miembros de la población. Otro método podría consistir en la eliminación en base a la *edad* de las cadenas.

Una de las claves para un buen rendimiento, tanto en la naturaleza como en los AGs, es mantener diversidad en la población el mayor tiempo posible. El efecto de la selección es la reducción de la diversidad y algunos métodos pueden reducirla con demasiada rapidez. Esto puede ser mitigado teniendo poblaciones muy numerosas o altas tasas de mutación, pero también existen otras técnicas.

Un enfoque popular, que generalmente se relaciona con los AGs de estado estacionario o incrementales, es usar una política de *no duplicidad*. Esto significa que la descendencia no se permite en la población si no son más que clones de los individuos existentes. La desventaja es la necesidad de comparar cada individuo existente con el nuevo candidato, lo que supone un esfuerzo computacional importante en grandes poblaciones. Podemos, por supuesto, tomar medidas para reducir la posibilidad de clonación antes de que los descendientes sean generados. Por ejemplo, antes de aplicar el cruce, podríamos examinar los padres seleccionados para encontrar los puntos de cruce adecuados.

1.2.2 Búsqueda dispersa

Desde el punto de vista de la clasificación de metaheurísticas, la Búsqueda Dispersa (SS por sus siglas en inglés, Scatter Search) puede ser vista como un algoritmo evolutivo basado en poblaciones que construye soluciones a través de la combinación de otras. Sus fundamentos se derivan de las estrategias propuestas originalmente para tratar reglas de decisión y restricciones en el contexto de programación entera.

SS ha demostrado proporcionar resultados prometedores para la resolución de problemas de optimización combinatoria y no lineales, utilizando estrategias eficaces para combinar vectores de soluciones en gran variedad de entornos problemáticos [11].

La Búsqueda Dispersa está diseñada para operar en un conjunto de puntos, llamados puntos de referencia, que constituyen buenas soluciones obtenidas a partir de soluciones anteriores. El enfoque genera sistemáticamente combinaciones de los puntos de referencia para crear nuevos puntos, siendo las combinaciones formas generalizadas de combinaciones lineales acompañadas de procesos para hacer cumplir las condiciones de factibilidad.

Algoritmo 2 - Esquema básico de búsqueda dispersa

```
1  Hacer P = ∅.
2  while (|P| < Psize) do
3      Utilizar el método de generación de diversificación para construir una solución x
4      if (x ∉ P) then
5          P = P ∪ {x}
6      else
7          Descartar x
8      end if
9  end while
10 Construir RefSet = {x^1, ..., x^b} con b soluciones diversificadas de P
11 Evaluar y ordenar de acuerdo al valor de su fitness las soluciones en RefSet, tal que x^1 es
   la mejor solución y x^b la peor
12 NuevasSoluciones = TRUE
13 while (NuevasSoluciones) do
14     Generar NuevosSubcjos, consistente en todos los pares de soluciones en RefSet
15     NuevasSoluciones = FALSE
16     while (NuevosSubcjos ≠ ∅) do
17         Seleccionar el siguiente subconjunto S de NuevosSubcjos
18         Aplicar el método de combinación de soluciones a S para obtener
           una o más nuevas soluciones x
19         if ((x ∉ RefSet) and (fitness de x es mejor que fitness de x^b)) then
20             Hacer x^b = x
21             Reordenar RefSet
22             NuevasSoluciones = TRUE
23         end if
24         Borrar S de NuevosSubcjos
25     end while
26 end while
```

El proceso de SS se organiza para capturar la información no contenida por

separado en los puntos originales, aprovechando métodos heurísticos de resolución auxiliares (para evaluar las combinaciones producidas y generar nuevos puntos) y haciendo uso de la propia estrategia en lugar de llevar a cabo las etapas del proceso al azar. SS consiste básicamente en cinco métodos [11]:

1. Un *método de generación de diversificación* para generar una colección de soluciones iniciales, utilizando una o más soluciones iniciales arbitrarias (o soluciones semilla) como entrada.

2. Un *método de mejora* para transformar una solución inicial en una o más soluciones iniciales mejoradas (ni las soluciones de entrada ni las de salida están obligadas a ser factibles, aunque las de salida suelen serlo). Si la solución inicial de entrada no mejora como resultado de la aplicación de este método, la solución "mejorada" se considera que es la misma que la solución de entrada.

3. Un *método de actualización del conjunto de referencia* para construir y mantener un conjunto de referencia formado por las b "mejores" soluciones encontradas (donde el valor de b suele ser pequeño, por ejemplo, no más de 20), organizado para proporcionar acceso eficiente a otras partes del procedimiento. Varios criterios alternativos pueden ser utilizados para añadir y eliminar soluciones al conjunto de referencia.

4. Un *método de generación de subconjuntos* para operar en el conjunto de referencia, produciendo un subconjunto de sus soluciones como base para la creación de soluciones combinadas. El método de generación de subgrupos más común es el de generar todos los pares de soluciones de referencia (es decir, todos los subconjuntos de tamaño 2).

5. Un *método de combinación de soluciones* para transformar un subconjunto dado de soluciones producidas por el método de generación de subconjuntos en una o más soluciones combinadas. El método de combinación es análogo al operador de cruce en algoritmos genéticos aunque debe ser capaz de combinar más de dos soluciones.

El procedimiento básico mostrado en el algoritmo 2 comienza con la creación de un conjunto de referencia inicial de soluciones (*RefSet*). El método de generación de diversificación se utiliza para construir un amplio conjunto de soluciones P. El tamaño de P (*Psize*) es típicamente 10 veces el tamaño de *RefSet*. Inicialmente, el conjunto de referencia *RefSet* se compone de las b soluciones distintas más diversas de P. Las soluciones en *RefSet* se ordenan de acuerdo a la calidad, donde la mejor solución es la primera en la lista. La búsqueda se inicia a continuación mediante la

asignación del valor *TRUE* a la variable booleana *NuevasSoluciones*. En las líneas 14 y 15, se construye *NuevosSubcjos* y *NuevasSoluciones* se cambia a *FALSE*. Nos centramos en subconjuntos de tamaño 2 por lo que el cardinal de *NuevosSubcjos* correspondientes al conjunto de referencia inicial viene dado por $(b^2 - b)/2$, que tiene en cuenta todos los pares de soluciones en *RefSet* (en subconjuntos de mayor tamaño son necesarias condiciones especiales para asegurar que se consigue una composición adecuada sin generar más de un número determinado de subconjuntos). Los pares en *NuevosSubcjos* se seleccionan uno cada vez en orden lexicográfico y el método de combinación de soluciones se aplica para generar una o más soluciones (línea 18). Si una solución recién creada mejora la peor de las soluciones actualmente en *RefSet*, la nueva solución sustituye a la peor y *RefSet* se reordena (líneas 20 y 21). La variable booleana *NuevasSoluciones* se cambia a *TRUE* y el subconjunto S que se ha combinado es eliminado de *NuevosSubcjos* (líneas 22 y 24). Este diseño básico puede ser ampliado y mejorado de diferentes maneras. La metodología SS es muy flexible, ya que cada uno de sus elementos pueden ser implementados en una variedad de formas y grados de sofisticación.

2

Algoritmos genéticos multiobjetivo

La presencia de múltiples objetivos en un problema da lugar, en principio, a un conjunto de soluciones eficientes. En ausencia de otra información adicional, una de estas soluciones eficientes no se puede decir que sea mejor que otra, lo que requiere encontrar el mayor número posible de estas soluciones. Los métodos clásicos de optimización multicriterio sugieren convertir el problema de optimización multiobjetivo en un problema de optimización con un solo objetivo, haciendo hincapié en una solución eficiente en particular cada vez. Cuando este método se utiliza para encontrar múltiples soluciones, tiene que ser aplicado muchas veces, con la esperanza de encontrar una solución diferente en cada ejecución.

En los últimos años se han desarrollado una serie de algoritmos evolutivos multiobjetivo (AEMOs) debido principalmente a su capacidad para encontrar múltiples soluciones eficientes en una sola ejecución. Dado que los algoritmos evolutivos trabajan con una población de soluciones, éstos pueden ser extendidos para mantener un conjunto diverso de soluciones. Con la finalidad de avanzar hacia la verdadera región óptima de Pareto, un algoritmo evolutivo se puede utilizar para encontrar múltiples soluciones Pareto-óptimas en una sola ejecución.

Estos algoritmos evolutivos (AE) desarrollados para resolver problemas de optimización multiobjetivo pusieron de manifiesto la necesidad de operadores adicionales para convertir un simple AE en un AEMO. Dos características comunes a todos ellos eran la asignación de un *fitness* a los miembros de la población basado en una

ordenación de soluciones no dominadas y la conservación de la diversidad entre las soluciones del mismo frente no dominado. Además, con el tiempo aumentó el interés por la introducción de elitismo para mejorar las propiedades de convergencia de un AEMO.

Por lo tanto, aproximar el frente de Pareto óptimo involucra dos objetivos (posiblemente en conflicto): la distancia al frente óptimo tiene que ser minimizada y la diversidad de las soluciones generadas tiene que ser maximizada. En este contexto, hay dos cuestiones fundamentales a la hora de diseñar un AEMO: cómo orientar la búsqueda hacia el frente óptimo de Pareto asignando valores de *fitness* a los individuos que permitan seleccionar cuáles se utilizarán para la generación de la descendencia, y qué individuos mantener durante el proceso de evolución, puesto que, debido a limitaciones de tiempo y recursos de almacenamiento, sólo una cierta fracción de los individuos de una generación concreta puede ser copiada en la siguiente generación. En la mayoría de los AEMO más modernos estas dos cuestiones se abordan de la siguiente manera, aunque los detalles pueden ser diferentes en cada uno de ellos:

Aptitud de los individuos: el grupo de individuos en cada generación se evalúa en dos etapas. En primer lugar, todos los individuos se comparan de acuerdo al concepto de dominancia, que define un orden parcial en el conjunto. La información de qué individuos dominan otro individuo, cuáles son los individuos que lo dominan o si son *indiferentes*, se utiliza para definir un ranking en la generación. Posteriormente, este ranking se refina mediante la información proporcionada por técnicas de estimación de la densidad, que miden la aglomeración de puntos en el lugar en el que se encuentra cada individuo específico.

Archivo de individuos: además de la población, se mantiene un archivo que contiene una aproximación del frente de Pareto, formado por puntos de entre todas las soluciones consideradas hasta el momento. Un miembro del archivo sólo se elimina si se encuentra otra solución que lo domina o si se supera el tamaño máximo del archivo y la parte del frente donde se encuentra este miembro del archivo está masificada. Por lo general, ser copiado en el archivo es la única manera de que un individuo pueda sobrevivir varias generaciones, por lo que esta técnica permite no perder ciertas porciones del frente no dominado por efectos aleatorios.

El estudio realizado en este trabajo se ha basado en el análisis del funcionamiento de tres importantes AEMO: NSGA-II, SPEA2 y AbYSS. A continuación, pasamos a detallar las principales características de cada uno de ellos. La nomenclatura utilizada en las siguientes secciones será la siguiente:

k	número de objetivos.
P	población (P_t: población en la t-ésima iteración del algoritmo).
N	tamaño de la población P.
$\bar{P}$	archivo ($\bar{P}_t$: archivo con el que se trabaja en la t-ésima iteración del algoritmo).
$\bar{N}$	tamaño del archivo $\bar{P}$.
F	$= (F_1, F_2, ...)$, conjunto de *frentes no dominados* que se obtiene de la población P.
M	tamaño del conjunto que se ofrece como aproximación del frente de Pareto.

2.1 NSGA-II

El algoritmo NSGA (Nondominated Sorting Genetic Algorithm) planteado en [22] fue uno de los primeros algoritmos evolutivos en aparecer. Con los años, las principales críticas a este algoritmo han sido la alta complejidad computacional en la ordenación de soluciones no dominadas ($O(kN^3)$), la falta de elitismo (el elitismo puede acelerar significativamente el rendimiento de un AG, a la vez que puede ayudar a prevenir la pérdida de buenas soluciones encontradas) y la necesidad de especificar un parámetro de relación entre soluciones, σ_{share}, para garantizar la diversidad en una población.

El algoritmo genético NSGA-II [5] es un AEMO basado en la ordenación de soluciones no dominadas que suaviza los problemas anteriores; en concreto, utiliza un enfoque de ordenación rápida de soluciones no dominadas de complejidad computacional $O(kN^2)$ y hace uso de un operador de selección que combina las poblaciones de padres e hijos y selecciona las mejores N soluciones (con respecto al *fitness* y al *spread*).

2.1.1 Ordenación rápida de soluciones no dominadas

En un primer enfoque, con el fin de identificar las soluciones del *primer frente no dominado* en una población de tamaño N, podría pensarse en comparar cada solución con cada una de las otras soluciones en la población para comprobar si está dominada. Esto requeriría $O(kN)$ comparaciones para cada solución. Si este proceso continuara para encontrar todos los miembros del primer nivel no dominado en la población, la complejidad total sería de $O(kN^2)$. En esta etapa, se habrían

Algoritmo 3 - Ordenación rápida de soluciones no dominadas

```
1  for each p ∈ P
2      S_p = ∅
3      n_p = 0
4      for each q ∈ P
5          if (p ≺ q) then
6              S_p = S_p ∪ {q}
7          else if (q ≺ p) then
8              n_p = n_p + 1
9          end if
10     end for
11     if n_p = 0 then
12         p_rank = 1
13         F_1 = F_1 ∪ {p}
14     end if
15 end for
16 i = 1
17 while F_i ≠ ∅ do
18     Q = ∅
19     for each p ∈ F_i
20         for each q ∈ S_p
21             n_p = n_p − 1
22             if n_p = 0 then
23                 q_rank = i + 1
24                 Q = Q ∪ {q}
25             end if
26         end for
27     end for
28     i = i + 1
29     F_i = Q
30 end while
```

encontrado todos los individuos en el primer frente no dominado. Con el fin de encontrar los individuos en el *siguiente frente no dominado*, las soluciones en el primer frente se descartarían temporalmente y el procedimiento anterior se repetiría. En el peor de los casos, la tarea de encontrar el segundo frente también requeriría $O(kN^2)$ cálculos, en particular cuando $O(N)$ número de soluciones pertenecen al segundo y superiores niveles no dominados. Este argumento seguiría siendo cierto para la búsqueda del tercer nivel y superiores; por lo tanto, el peor caso se daría cuando hubiera N frentes y sólo existiera una solución en cada frente. Esto requeriría un total de $O(kN^3)$ cálculos. Nótese que se requeriría almacenamiento del orden de $O(N)$ para este procedimiento.

Como se ha comentado, en [5] se propone un algoritmo de ordenación rápida de soluciones no dominadas que requiere $O(kN^2)$ cálculos y que se muestra en el algoritmo 3.

En primer lugar, para cada solución p se calculan el número de soluciones que dominan a p, n_p, y el conjunto de soluciones a las que p domina, S_p (líneas 5-9). Esto requiere $O(kN^2)$ comparaciones. Todas las soluciones del primer frente no dominado tendrán su contador n_p a cero (línea 11). Cada solución p indica a qué frente pertenece a través del atributo p_{rank} (líneas 12 y 23). A continuación, para cada solución con $n_p = 0$, visitamos a cada miembro q de su conjunto S_p y reducimos su contador n_q en uno (línea 21). De este modo, si para cualquier miembro de la población q, su contador n_q toma el valor cero, q se sitúa en una lista separada Q (línea 24). Estos miembros pertenecen al segundo frente no dominado. Después, el procedimiento anterior se repite con cada miembro de Q y se identifica el tercer frente. Este proceso continúa hasta que todos los frentes son identificados.

Para cada solución p en el segundo nivel o superiores, su contador n_p puede ser, a lo sumo, $N - 1$. Por lo tanto, cada solución será visitada como mucho $N - 1$ veces antes de que su contador tome el valor cero. En ese momento, la solución es asignada a un nivel no dominado y nunca vuelve a ser visitada. Puesto que hay a lo sumo $N - 1$ de tales soluciones, la complejidad total es $O(N^2)$, por lo que la complejidad global del procedimiento es $O(kN^2)$. Es importante tener en cuenta que, aunque la complejidad en cuanto a tiempo se reduce a $O(kN^2)$, el requisito de almacenamiento aumenta a $O(N^2)$.

2.1.2 Conservación de la diversidad

Junto con la convergencia al frente de Pareto óptimo, también es deseable que un algoritmo evolutivo mantenga una buena diversidad o *spread* en las soluciones del frente no dominado que ofrece como solución. El algoritmo NSGA original utiliza un enfoque que hace uso del parámetro σ_{share} anteriormente comentado y que está relacionado con la distancia elegida para calcular la proximidad entre dos miembros de la población. El parámetro σ_{share} denota el mayor valor de esta distancia dentro de la cual dos soluciones cualesquiera pueden compartir su *fitness*. Este enfoque proporciona una buena diversidad en la población con la configuración adecuada de los parámetros asociados, pero presenta dos problemas: el rendimiento del método depende en gran medida del valor elegido para σ_{share} y, puesto que cada solución debe ser comparada con el resto de soluciones en la población, la complejidad global

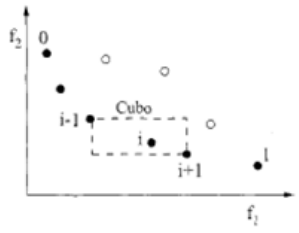

Figura 2.1: Cálculo de la *crowding-distance*. Los puntos negros son las soluciones del mismo frente no dominado.

es de $O(N^2)$.

En el algoritmo NSGA-II se utiliza un enfoque de comparación de la concentración de soluciones que elimina, hasta cierto punto, las dificultades anteriores. El nuevo enfoque no requiere ningún parámetro definido por el usuario para mantener la diversidad entre los miembros de la población y, además, tiene una complejidad computacional mejor.

En primer lugar, necesitamos definir el *estimador de densidad*. Para obtener una estimación de la densidad de soluciones que rodean a una solución en particular de la población en el espacio imagen, se calcula la distancia entre los puntos a cada lado de este punto en cada uno de los objetivos. La suma de dichas distancias, $i_{distance}$, sirve como una estimación del perímetro del cubo formado usando los vecinos más cercanos como vértices, y se llama *crowding-distance*. En la figura 2.1, la *crowding-distance* de la i-ésima solución en su frente (marcado con círculos negros) es la longitud media de los lados del cubo (mostrado con líneas discontinuas).

El algoritmo 4 describe el procedimiento de cálculo de la *crowding-distance* de todas las soluciones en un conjunto I de puntos no dominados. Este cálculo requiere la ordenación de los puntos de dicho conjunto de acuerdo a cada valor de la función objetivo de forma ascendente en orden de magnitud (línea 6). Después, para cada función objetivo, a las soluciones límite (soluciones con menor y mayor valor objetivo) se les asigna un valor distancia infinito (línea 7). A todas las demás soluciones intermedias se les asigna un valor distancia igual a la diferencia absoluta normalizada de los valores de la función en las dos soluciones adyacentes, y este cálculo se realiza con todas las funciones objetivo. El valor general de la *crowding-distance* se calcula como la suma de los valores de las distancias individuales correspondientes a cada uno de los objetivos (línea 9) donde cada función objetivo es normalizada antes de calcular esta distancia.

Algoritmo 4 - Asignación de *crowding-distance*

```
1  l = |I|
2  for each i
3        I[i]_distance = 0
4  end for
5  for each objetivo k
6        I = ordenar(I, k)
7        I[1]_distance = I[l]_distance = ∞
8        for i = 2 to (l − 1)
9              I[i]_distance = I[i]_distance + (I[i + 1].k − I[i − 1].k)/(f_k^max − f_k^min)
10       end for
11 end for
```

$I[i].k$ se refiere al valor de la k-ésima función objetivo en el individuo i-ésimo del conjunto I y los parámetros f_k^{max} y f_k^{min} son los valores máximo y mínimo de la k-ésima función objetivo. La complejidad de este procedimiento se rige por el algoritmo de ordenación. Puesto que se realizan k ordenaciones independientes de, a lo sumo, N soluciones (cuando todos los miembros de la población se encuentran en un solo frente I), el algoritmo anterior tiene una complejidad de $O(kN \log N)$.

Cuando todos los miembros de la población en el conjunto I tienen asignada una distancia, podemos comparar dos soluciones por su grado de proximidad con otras soluciones. Una solución con menor valor de esta medida de distancia estará, en cierto sentido, más apiñada por otras soluciones. Esto es exactamente lo que se compara en el operador propuesto en el algoritmo NSGA-II, el *crowded-comparison operator*. Aunque la figura 2.1 ilustra el cálculo de la *crowding-distance* para dos objetivos, el procedimiento también es aplicable a más de dos objetivos.

El *crowded-comparison operator* comentado ($\prec_n$) guía el proceso de selección en las diversas etapas del algoritmo hacia un frente de Pareto óptimo distribuido de manera uniforme. En NSGA-II, cada individuo de la población i tiene dos atributos: rango no-dominado (i_{rank}) y *crowding-distance* ($i_{distance}$). El algoritmo define un orden parcial $\prec_n$ como:

$$i \prec_n j \text{ si } (i_{rank} < j_{rank}) \text{ o } ((i_{rank} = j_{rank}) \text{ y } (i_{distance} > j_{distance})).$$

Es decir, entre dos soluciones con diferentes rangos es preferible la solución con el menor (mejor) rango. En otro caso, si ambas soluciones pertenecen al mismo frente, entonces es preferible la solución que se encuentra en una región menos *densa*.

2.1.3 Algoritmo NSGA-II

Conociendo los procedimientos de ordenación rápida de soluciones no dominadas y de estimación rápida de la *crowding-distance* y el *crowded-comparison operator* se puede pasar a describir el algoritmo NSGA-II.

Inicialmente, se crea una población al azar P_0 y se ordena según los diferentes frentes no dominados (a cada solución se le asigna como *fitness* su rango no-dominado: 1 es el mejor rango, 2 es el siguiente mejor rango y así sucesivamente). En un primer momento, se hace uso de los operadores de selección por torneo binario, cruce y mutación para crear una población de descendientes Q_0 de tamaño N. Puesto que se introduce el elitismo comparando la población actual con las mejores soluciones no-dominadas encontradas con anterioridad, el procedimiento es diferente después de la generación de la población inicial. En el algoritmo 5 se describe la construcción de la t-ésima generación.

Algoritmo 5 - Construcción de la t-ésima generación

```
1  R_t = P_t ∪ Q_t
2  F = ordenación-rápida-soluciones-nodominadas(R_t)
3  P_{t+1} = ∅
4  i = 1
5  while |P_{t+1}| + |F_i| ≤ N do
6        asignación-crowding-distance(F_i)
7        P_{t+1} = P_{t+1} ∪ F_i
8        i = i + 1
9  end while
10 ordenar(F_i, ≺_n)
11 P_{t+1} = P_{t+1} ∪ F_i[1 : (N − |P_{t+1}|)]
12 Q_{t+1} = crear-nueva-población(P_{t+1})
13 t = t + 1
```

En primer lugar, se crea una población R_t, de tamaño $2N$, formada por la combinación de la población de la iteración anterior, P_t, y su conjunto de descendientes recién creados, Q_t (línea 1), ambos de tamaño N. A continuación, la población R_t se ordena de acuerdo a sus rangos no dominados (línea 2, donde $F = (F_1, F_2, ...)$ son todos los frentes no dominados de R_t). El elitismo está garantizado, puesto que todos los miembros de la población actual y de la anterior están incluidos en R_t. Las soluciones pertenecientes a F_1 son las mejores soluciones en la población combinada R_t, por lo que hay que darles mayor importancia que a otras soluciones de dicho conjunto. Si el tamaño de F_1 es menor que N, se eligen todos los miembros de ese

conjunto para la nueva población P_{t+1} y los restantes miembros se eligen de los siguientes frentes no dominados de acuerdo a su rango (las soluciones del conjunto F_2 se eligen a continuación, seguidas por las soluciones de F_3 y así sucesivamente). Este procedimiento continúa hasta que no pueden añadirse más frentes no dominados completos.

Sea F_l el último conjunto no dominado añadido a la nueva población P_{t+1}, en general, el número de soluciones en todos los conjuntos desde F_1 hasta F_l será mayor que el tamaño de la población, por lo que, para elegir exactamente N miembros de la población, se ordenan las soluciones del último frente F_l usando el *crowded-comparison operator* $\prec_n$ en orden descendente (línea 10) y se eligen las mejores soluciones necesarias para completar la población (línea 11). A la nueva población P_{t+1} de tamaño N se le aplican entonces los operadores de selección, cruce y mutación para crear una nueva población Q_{t+1} de tamaño N (línea 12). Es importante observar que se utiliza el operador de selección por torneo binario pero el criterio de selección se basa en el *crowded-comparison operator* $\prec_n$. Puesto que este operador requiere tanto del rango como de la *crowding-distance* de cada solución de la población, se calculan estas cantidades a la vez que se forma la población P_{t+1}.

Si tenemos en cuenta una iteración del algoritmo, las operaciones básicas y sus complejidades en el peor de los casos serían [5]: $O(k(2N)^2)$ para la ordenación de soluciones no dominadas, $O(k(2N)log(2N))$ para la asignación de la *crowding-distance* y $O(2Nlog(2N))$ para la ordenación con $\prec_n$. Por lo que la complejidad global del algoritmo es $O(kN^2)$, gobernado por la parte de ordenación de soluciones no dominadas del algoritmo. Obsérvese que la población de tamaño $2N$ no necesita ser ordenada por completo de acuerdo a criterios de no dominancia, puesto que tan pronto como el procedimiento de ordenación haya encontrado un número suficiente de frentes para obtener N miembros en P_{t+1} no hay razón para continuar con la ordenación. La diversidad entre las soluciones no dominadas se introduce mediante el uso del *crowded-comparison operator*, que se utiliza en la selección por torneo y durante la fase de reducción de la población. Puesto que las soluciones compiten con su *crowding-distance* (una medida de la densidad de las soluciones en su entorno), no se requieren parámetros adicionales como es el caso del σ_{share} en el algoritmo NSGA original.

2.2 SPEA2

El algoritmo SPEA (Strength Pareto Evolutionary Algorithm) (Zitzler y Thiele 1999) es una técnica relativamente reciente para encontrar o aproximar el frente de Pareto en problemas de optimización multiobjetivo. En este trabajo se ha analizado el funcionamiento del algoritmo SPEA2 [27], una versión mejorada que integra, en contraste con su predecesor, una estrategia refinada de asignación de *fitness*, una técnica de estimación de densidad y un método mejorado de truncamiento del archivo.

Como SPEA es la base para SPEA2, haremos un breve resumen de este algoritmo [27]. SPEA usa una población ordinaria y un archivo (un conjunto externo). Comenzando con una población inicial y un archivo vacío, se realizan los pasos siguientes en cada iteración. En primer lugar, todos los miembros de la población no dominados se copian en el archivo; cualquier individuo dominado o duplicado (con respecto a los valores objetivos) se elimina del archivo durante esta operación de actualización. Si el tamaño del archivo actualizado supera un límite predefinido, se eliminan algunos miembros del archivo mediante una técnica de agrupación que conserva las características del frente no dominado. A continuación, se asignan valores de *fitness* tanto a los individuos del archivo como a los miembros de la población:

- A cada individuo i en el archivo se le asigna un valor de importancia $S(i) \in [0, 1)$ que, al mismo tiempo, representa su valor de *fitness* $F(i)$. $S(i)$ es el número j de miembros de la población que están dominados o son iguales que i con respecto a los valores objetivo, dividido por el tamaño de la población más uno.

- El *fitness* $F(j)$ de un individuo j en la población se calcula sumando los valores $S(i)$ de todos los miembros i del archivo que dominan o son iguales a j, más uno.

El siguiente paso consiste en la selección por torneo binario de la unión de los individuos de la población y del archivo. Nótese que cuanto menor es el *fitness* mejor, así que cada individuo en el archivo tiene una mayor posibilidad de ser seleccionado que cualquier miembro de la población. Finalmente, después del cruce y la mutación, la población se sustituye por su descendencia.

A pesar de que SPEA ha obtenido buenos resultados en diferentes estudios [25, 26], se han identificado algunas debilidades tales como:

1. Asignación de *fitness*: los individuos que son dominados por los mismos miembros del archivo tienen los mismos valores de *fitness*. Esto significa que en el caso de que el archivo contenga un solo individuo, todos los miembros de la

población tienen el mismo rango independiente de si se dominan entre sí o no. Como consecuencia, la presión de selección se reduce sustancialmente y, en este caso en particular, SPEA se comporta como un algoritmo de búsqueda aleatoria.

2. Estimación de la densidad: si muchos individuos de la generación actual son indiferentes, es decir, no se dominan entre ellos, se puede obtener muy poca información en base al orden parcial definido por la relación de dominancia. En esta situación, que es muy probable que ocurra con la presencia de más de dos objetivos, la información de densidad tiene que ser utilizada para guiar la búsqueda de una forma más eficaz.

3. Truncamiento del archivo: aunque la técnica de agrupamiento utilizado en SPEA es capaz de reducir el conjunto no dominado sin destruir sus características, puede perder las soluciones más alejadas. Sin embargo, estas soluciones se deben mantener en el archivo para obtener una buena distribución de soluciones no dominadas.

En contraste con el algoritmo SPEA, SPEA2 utiliza una estrategia refinada de asignación de *fitness* que incorpora información de densidad. Además, el tamaño del archivo es fijo, es decir, cuando el número de individuos no dominados es menor que el tamaño predefinido del archivo, el archivo se completa con individuos dominados (con SPEA el tamaño del archivo puede variar con el tiempo). Respecto a la técnica de truncamiento, utilizada cuando el frente no dominado supera el límite del archivo, es reemplazada por un método similar pero que no pierde puntos límite. Otra diferencia con respecto a SPEA es que sólo los miembros del archivo participan en el proceso de selección.

2.2.1 *Fitness* y estimación de densidad

Para evitar que los individuos dominados por los mismos miembros del archivo tengan valores de *fitness* idénticos, con SPEA2 para cada individuo son tenidas en cuenta tanto las soluciones que domina como aquellas por las que es dominado. A cada individuo i en el archivo $\bar{P}_t$ y en la población P_t se le asigna un valor de importancia $S(i)$, que representa el número de soluciones que domina:

$$S(i) = |\{j | j \in P_t \cup \bar{P}_t \wedge i \succ j\}| \tag{2.1}$$

Haciendo uso de los valores de S, se obtiene un primer valor de *fitness sin refinar* $R(i)$ para un individuo i, dado por los valores de $S(j)$ de los individuos j que dominan a i tanto en el archivo como en la población (en oposición a SPEA donde sólo son considerados los miembros del archivo):

$$R(i) = \sum_{j \in P_t \cup \bar{P}_t, j \succ i} S(j) \tag{2.2}$$

Es importante señalar que cuanto menor es el *fitness*, mejor. Así, $R(i) = 0$ corresponde a un individuo no dominado, mientras que un valor alto de $R(i)$ significa que i está dominado por muchos individuos (que a su vez dominan a muchos otros).

Obsérvese que este primer valor de *fitness* puede fallar cuando la mayoría de individuos no se dominan entre sí. Por lo tanto, se incorpora información adicional de densidad para discriminar entre individuos que tengan idénticos valores de $R(i)$. La técnica de estimación de la densidad utilizada en SPEA2 es una adaptación del método del r-ésimo vecino más cercano de [21], donde la densidad en cualquier punto es una función (decreciente) de la distancia al r-ésimo punto más cercano. En SPEA2, se toma la inversa de la distancia al r-ésimo vecino más cercano como estimación de la densidad. Para cada individuo i, se calculan las distancias (en el espacio objetivo) a todos los individuos j en el archivo y en la población, y se almacenan en una lista. Tras ordenar la lista en orden creciente, el elemento r-ésimo informa de la distancia buscada, denotada por σ_i^r. La r utilizada en SPEA2 es igual a la raíz cuadrada del tamaño de la muestra [21], por lo tanto, $r = \sqrt{N + \bar{N}}$, donde N es el tamaño de la población y $\bar{N}$ el tamaño del archivo. Con todo esto, la densidad $D(i)$ correspondiente al individuo i se define por:

$$D(i) = \frac{1}{\sigma_i^r + 2} \tag{2.3}$$

En el denominador se suma 2 para asegurar que su valor es mayor que cero y que $D(i) < 1$. Finalmente, sumando $D(i)$ al valor de $R(i)$ se obtiene el valor de *fitness* $F(i)$ de un individuo i:

$$F(i) = R(i) + D(i) \tag{2.4}$$

El tiempo de ejecución del procedimiento de asignación de *fitness* está dominado por el estimador de densidad ($O(L^2 logL)$), mientras que el cálculo de los valores de S y R tiene una complejidad de $O(L^2)$, donde $L = N + \bar{N}$.

2.2.2 Archivo

La operación de actualización del archivo en SPEA2 difiere de la utilizada en SPEA en dos aspectos: el número de individuos en el archivo es constante y el método de truncamiento evita que las soluciones más extremas sean eliminadas. El primer paso consiste en copiar todos los individuos no dominados, es decir, aquellos que tienen un *fitness* menor que uno, del archivo y de la población al archivo de la próxima generación:

$$\bar{P}_{t+1} = \{i | i \in P_t \cup \bar{P}_t \wedge F(i) < 1\} \tag{2.5}$$

Si el frente no dominado encaja exactamente en el archivo ($|\bar{P}_{t+1}| = \bar{N}$) la operación queda completada. De lo contrario, pueden darse dos situaciones, o bien el archivo es demasiado pequeño o bien demasiado grande. En el primer caso, $|\bar{P}_{t+1}| < \bar{N}$, los mejores $\bar{N} - |\bar{P}_{t+1}|$ individuos dominados en el archivo y población anteriores se copian al nuevo archivo. Esto puede ser implementado ordenando el conjunto $P_t \cup \bar{P}_t$ de acuerdo a los valores de *fitness* y copiando a $\bar{P}_{t+1}$ los primeros $\bar{N} - |\bar{P}_{t+1}|$ individuos i de esa lista ordenada con $F(i) > 1$. En el segundo caso, $|\bar{P}_{t+1}| > \bar{N}$, se invoca un procedimiento de truncamiento del archivo que iterativamente elimina individuos de $\bar{P}_{t+1}$ hasta que $|\bar{P}_{t+1}| = \bar{N}$. En este caso, en cada iteración se elige para eliminar el individuo i para el que $i \leq_d j$ para todo $j \in \bar{P}_{t+1}$, con

$$\begin{array}{lcll} i \leq_d j & \Leftrightarrow & \forall\ 0 < r < |\bar{P}_{t+1}| : & \sigma_i^r = \sigma_j^r \quad \vee \\ & & \exists\ 0 < r < |\bar{P}_{t+1}| : & [(\forall\ 0 < l < r : \ \sigma_i^l = \sigma_j^l) \wedge \sigma_i^r < \sigma_j^r] \end{array} \tag{2.6}$$

donde σ_i^r denota la distancia de i a su r-ésimo vecino más cercano en $\bar{P}_{t+1}$. En otras palabras, se elige el individuo que tiene la distancia mínima a otro individuo, y si hay varios individuos con distancia mínima se rompe el empate considerando las segundas o terceras distancias más pequeñas y así sucesivamente. La figura 2.2 ilustra el funcionamiento de esta técnica de truncamiento.

Aunque, el peor tiempo de ejecución del operador de truncamiento es $O(L^3)$ [27], ($L = N + \bar{N}$), en promedio la complejidad será menor ($O(L^2 log L)$) puesto que los individuos generalmente difieren con respecto al segundo o tercer vecino más cercano, y entonces la ordenación de las distancias gobernaría la complejidad global.

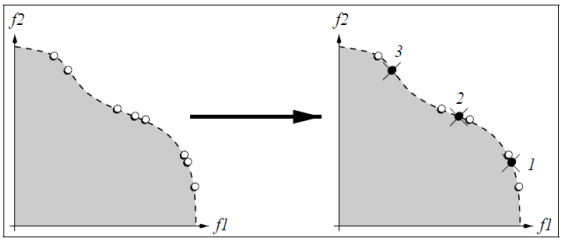

Figura 2.2: Método de truncamiento de archivo en SPEA2. A la derecha, un conjunto no dominado. A la izquierda, las soluciones (en orden) que son eliminadas ($\bar{N} = 5$).

2.2.3 Algoritmo SPEA2

Con todo lo visto, el esquema general de SPEA2 sería como muestra el algoritmo 6, donde T es el número máximo de generaciones.

Algoritmo 6 - Algoritmo SPEA2 - Esquema general

```
1  Generar una población inicial P_0
2  Crear un archivo vacío (conjunto externo) P̄_0 = ∅
3  t = 0
4  while t < T o no se satisfaga otro criterio de parada do
5      Calcular los valores de fitness de los individuos en P_t y P̄_t
6      Copiar todos los individuos no dominados en P_t y P̄_t a P̄_{t+1}
7      if |P̄_{t+1}| > N̄ then
8          Reducir P̄_{t+1} por medio del operador de truncamiento
9      else if |P̄_{t+1}| < N̄ then
10         Completar P̄_{t+1} con individuos dominados en P_t y P̄_t
11     end if
12     Aplicar el operador de selección por torneo binario con reemplazamiento en P̄_{t+1} hasta
       completar el grupo de individuos que se reproducirán.
13     Aplicar los operadores de cruce y mutación y establecer en P_{t+1} la población resultante
14     t = t + 1
15 end while
16 Devolver P̄_t
```

2.3 AbYSS

El algoritmo AbYSS (Archived-Based hYbrid Scatter Search) [17] adapta el esquema de la búsqueda dispersa (SS, por sus siglas en inglés) para la optimización

de un solo objetivo al dominio multiobjetivo. El resultado es una metaheurísitca híbrida que sigue la estructura de SS, pero utiliza los operadores de mutación y de cruce de los algoritmos evolutivos. AbYSS incorpora conceptos típicos del campo multiobjetivo, tales como la dominancia de Pareto, la estimación de densidad y un archivo externo para almacenar las soluciones no dominadas.

SS es una metaheurística que se puede considerar un algoritmo evolutivo en el sentido de que incorpora el concepto de población. Sin embargo, por lo general, SS evita el uso de muchos componentes aleatorios, y operadores evolutivos típicos como la mutación o cruce que no se ajustan, en teoría, con la filosofía de este algoritmo. El método se basa en el uso de una pequeña población, conocida como *conjunto de referencia*, cuyos individuos se combinan para construir nuevas soluciones que se generan de forma sistemática. Además, estos nuevos individuos pueden ser mejorados mediante la aplicación de un método de búsqueda local. El conjunto de referencia se inicializa a partir de una población inicial compuesta de diversas soluciones y se actualiza con las soluciones obtenidas en la búsqueda local.

Como se comentó en la sección 1.2.2, la estructura de SS se caracteriza por cinco métodos que deben definirse: método de generación de diversificación, método de mejora, método de actualización del conjunto de referencia, método de generación de subconjuntos y método de combinación de soluciones. En comparación con los algoritmos evolutivos, SS es atractivo debido a que es una estrategia estructurada, en la que se expresa claramente dónde pueden ser aplicadas las búsquedas locales (en el método de mejora), y donde el equilibrio entre diversificación e intensificación se puede ajustar de varias formas: ajustando el tamaño del conjunto de referencia, definiendo cómo se actualiza el conjunto de referencia (método de actualización del conjunto de referencia) y determinando cómo crear nuevas soluciones (en el método de generación de subconjuntos). Hasta hace poco, SS no se había ampliado para hacer frente a problemas de optimización multiobjetivo.

AbYSS utiliza un archivo externo para almacenar las soluciones no dominadas encontradas durante la búsqueda, combinando además aspectos de los algoritmos NSGA-II y SPEA2 que se acaban de tratar. Hace uso de la *crowding-distance* de NSGA-II como medida de diversidad y aplica la estimación de densidad del algoritmo SPEA2 en la selección de las soluciones del conjunto inicial utilizado para construir el conjunto de referencia.

2.3.1 Especificaciones de AbYSS

El método de búsqueda dispersa comienza con la creación de un conjunto inicial P de diversas soluciones en la fase de inicialización. Esta fase consiste en la generación de nuevas soluciones de forma iterativa mediante la invocación del método de generación de diversificación. A cada solución se le aplica el método de mejora, que por lo general es una búsqueda local, y el individuo resultante se añade al conjunto inicial. Después de esta fase inicial, comienza el bucle principal de SS.

El bucle principal empieza con la construcción del conjunto de referencia, $RefSet$, desde el conjunto inicial con el método de actualización del conjunto de referencia. Entonces, las soluciones en el conjunto de referencia son sistemáticamente agrupadas en subconjuntos de dos o más individuos haciendo uso del método de generación de subconjuntos. En el siguiente paso, las soluciones de cada subconjunto se combinan de alguna manera para producir nuevos individuos. Esta combinación está definida en el método de combinación de soluciones. El método de mejora se aplica a cada solución recién generada, y el paso final es decidir si la solución resultante se inserta o no en el conjunto de referencia. Este bucle se ejecuta hasta que se cumple una condición de parada (por ejemplo, se alcanza un número dado de iteraciones o el método de generación de subconjuntos no produce más subconjuntos). Opcionalmente, puede realizarse un proceso de reinicio. La idea es crear un nuevo conjunto inicial, que contiene los individuos actualmente en el conjunto de referencia, y los individuos restantes se generan mediante los métodos de generación de diversificación y de mejora, como en la fase inicial.

Se describe a continuación cómo son definidos cada uno de los cinco métodos de SS en el algoritmo AbYSS [17].

Método de generación de diversificación: el objetivo es generar un conjunto inicial P de soluciones diversas. Es un método sencillo basado en dividir el rango de cada variable en un número de subrangos de igual tamaño; a continuación, el valor de cada variable de decisión de cada solución se genera en dos pasos. En primer lugar, se elige un subrango de la variable aleatoriamente, donde la probabilidad de seleccionar un subrango es inversamente proporcional a su frecuencia (número de veces que el subrango ya ha sido seleccionado). En segundo lugar, se genera aleatoriamente un valor dentro del intervalo seleccionado, y este proceso se repite para todas las variables de decisión de todas las soluciones.

Método de mejora: la idea detrás de este método es utilizar un algoritmo de búsqueda local para mejorar las soluciones obtenidas con el método de generación

de diversificación y el método de combinación de soluciones. AbYSS utiliza una estrategia (1+1) de Algoritmos Evolutivos (AE). Esta estrategia, que ha demostrado un buen desempeño en AEs, se basa en utilizar un operador de mutación como perturbación y un test de dominancia de Pareto. El esquema de este método se muestra en el algoritmo 7.

Algoritmo 7 - Método de mejora

```
1  individuo Mejora(individuo IndividuoOriginal, int iter) {
2      individuo IndividuoMutado
3      repeat iter times
4          IndividuoMutado = Mutar(IndividuoOriginal)
5          if (el problema tiene restricciones) then
6              EvaluarRestricciones(IndividuoMutado)
7              mejor = TestRestricciones(IndividuoMutado, IndividuoOriginal)
8              if (ninguno es mejor que el otro) then
9                  Evaluar(IndividuoMutado)
10                 mejor = TestDominancia(IndividuoMutado, IndividuoOriginal)
11             else if (IndividuoMutado es mejor) then
12                 Evaluar(IndividuoMutado)
13             end if
14         else // (el problema no tiene restricciones)
15             Evaluar(IndividuoMutado)
16             mejor = TestDominancia(IndividuoMutado, IndividuoOriginal)
17         end if
18         if (IndividuoMutado es mejor) then
19             IndividuoOriginal = IndividuoMutado
20         else if (IndividuoOriginal es mejor) then
21             Eliminar(IndividuoMutado)
22         else if (IndividuoMutado no está dominado por el archivo) then
23             Insertar IndividuoOriginal en el archivo
24             IndividuoOriginal = IndividuoMutado
25         else
26             Eliminar(IndividuoMutado)
27         end if
28     end repeat
29     Devolver IndividuoOriginal
30 }
```

Al método de mejora se le pasa un individuo como parámetro, que es repetidamente mutado con el objetivo de obtener un individuo mejor (AbYSS utiliza un operador de mutación polinomial, típicamente utilizado por algoritmos genéticos multiobjetivo, tales como NSGA-II y SPEA2). Un test de violación de restricciones comprueba si los individuos son factibles o no, si uno de ellos lo es y el otro no, o

ambos son infactibles pero uno tiene una violación global menor, entonces el test devuelve el mejor (línea 7). De lo contrario, se utiliza una prueba de dominancia para decidir si uno de los individuos domina al otro. Si el individuo original es mejor, el mutado es descartado y si, por el contrario, el individuo mutado es mejor, éste reemplaza al original; por último, si ninguno domina al otro y el individuo mutado no está dominado por el archivo, el individuo original se mueve al archivo y el mutado se convierte en el nuevo original. De esta manera se evita empeorar soluciones con respecto al archivo a medida que avanza el proceso de mejora.

Es interesante señalar varias características del método de mejora propuesto en AbYSS. En primer lugar, no se pierde ninguna solución no dominada ya que, en el caso de que se encuentren varias soluciones no dominadas en el procedimiento, éstas son insertadas en el archivo. En segundo lugar, ajustando el parámetro *iter*, se puede ajustar fácilmente la intensificación.

Método de actualización del conjunto de referencia: el conjunto de referencia es una colección de soluciones diversas y de alta calidad que son utilizadas para generar nuevos individuos. El propio conjunto se compone de dos subconjuntos, $RefSet_1$ y $RefSet_2$, de tamaños p y q, respectivamente. El primer subconjunto contiene las mejores soluciones de P en cuanto a calidad, mientras que el segundo subconjunto debe contener soluciones que promuevan diversificación. En [16], el conjunto $RefSet_2$ se construye seleccionando de P aquellos individuos cuya distancia euclídea a $RefSet_1$ es la mayor. AbYSS utiliza la misma estrategia para construir $RefSet_2$ pero, como es habitual en la optimización multiobjetivo, es necesario definir el concepto de "mejor individuo" para construir $RefSet_1$. El método de actualización del conjunto de referencia también se utiliza para actualizar el conjunto de referencia con las nuevas soluciones obtenidas en el bucle principal de AbYSS. Un esquema de este método puede verse en el algoritmo 8.

Para seleccionar los mejores p individuos de P (línea 3), AbYSS utiliza el enfoque utilizado en SPEA2, es decir, a los individuos se les asigna un valor de *fitness* que es la suma de *fitness sin refinar* más una estimación de la densidad [27]. Recordemos que la importancia de un individuo es el número de soluciones que el individuo domina en la población y el *fitness sin refinar* es la suma de las importancias de los individuos que lo dominan. La estimación de la densidad se basa en el cálculo de la distancia al r-ésimo vecino más cercano.

Una vez que el conjunto de referencia se completa, sus soluciones se combinan para obtener nuevas soluciones que a continuación son mejoradas. Posteriormente, estas nuevas soluciones se testean para su inclusión en el conjunto de referencia (línea

Algoritmo 8 - Método de actualización del conjunto de referencia

```
1  ActualizacionConjuntoReferencia(bool build) {
2      if (build) then // (construir un nuevo conjunto de referencia)
3          Seleccionar los p mejores individuos de P
4          Construir RefSet_1 con esos p individuos
5          Calcular las distancias euclídeas en P para obtener q individuos
6          Construir RefSet_2 con esos q individuos
7      else // (actualizar el conjunto de referencia)
8          for each nueva solución s
9              Intentar insertar s en RefSet_1
10             if (no es posible insertarla) then
11                 Intentar insertar s en RefSet_2
12                 if (no es posible insertarla) then
13                     Borrar s
14                 end if
15             end if
16         end for
17     end if
18 }
```

7 del algoritmo 8). De acuerdo con el esquema de búsqueda dispersa para problemas uniobjetivo, una nueva solución puede convertirse en un miembro del conjunto de referencia si satisface una de las siguientes condiciones:

- El nuevo individuo tiene un valor de la función objetivo mejor que el individuo con peor valor objetivo en $RefSet_1$.

- El nuevo individuo tiene una distancia al conjunto de referencia mejor que el individuo con la peor distancia en $RefSet_2$.

Para la primera condición, y por tratarse de optimización multiobjetivo, es necesario especificar a qué nos referimos con el concepto de "mejor individuo". Para determinar si una nueva solución es mejor que otra en $RefSet_1$, es decir, la comprobación realizada para intentar insertar un nuevo individuo s en $RefSet_1$ de la línea 9 del algoritmo 8, no se puede utilizar un procedimiento de clasificación porque el tamaño de esta población suele ser pequeño (típicamente, el tamaño del conjunto de referencia es 20 o menos). El enfoque utilizado en AbYSS consiste en comparar cada nueva solución s con los individuos en $RefSet_1$ haciendo uso de un test de dominancia. Este test se muestra en el algoritmo 9 (por simplicidad, no se consideran restricciones en el problema de optimización multiobjetivo; si las hubiera el método a seguir sería como el indicado en el algoritmo 7).

Algoritmo 9 - Test para añadir un nuevo individuo s a $RefSet_1$

```
1  bool dominado = false
2  for each solución r en RefSet1
3       if (s domina a r) then
4            Borrar r de RefSet1
5       else if (r domina a s) then
6            bool dominado = true
7       end if
8       if (no dominado) then
9            if (RefSet1 no está lleno) then
10                Añadir s a RefSet1
11           else
12                Añadir s al archivo
13           end if
14      else // (el individuo s está dominado)
15           // (Intentar actualizar RefSet2 con el individuo s)
16           ...
```

Téngase en cuenta que cuando un nuevo individuo no está dominado por $RefSet_1$ se inserta en este conjunto sólo si no está lleno. Esto significa que el nuevo individuo tiene que dominar al menos a un individuo en $RefSet_1$. Si esta condición no se cumple, el individuo se inserta en el archivo.

Método de generación de subconjuntos: de acuerdo con el esquema de búsqueda dispersa, este método genera subconjuntos de individuos, que se utilizarán para crear nuevas soluciones con el método de combinación de soluciones. Son posibles varios tipos de subconjuntos aunque la estrategia más habitual consiste en considerar todas las combinaciones de pares de soluciones en el conjunto de referencia. Además, este método debería evitar la producción de subconjuntos repetidos de individuos, es decir, subconjuntos generados previamente.

En AbYSS este método produce, por un lado, parejas de combinaciones de individuos de $RefSet_1$ y, por otro lado, combinaciones de pares de individuos de $RefSet_2$. Experimentos realizados en [17] revelan que la generación de combinaciones de individuos de los dos subconjuntos hace que el algoritmo converja de un modo pobre. La razón está relacionada con el hecho de que la combinación de los individuos de los dos *RefSets* aumenta las capacidades de exploración de la búsqueda, lo que produce un desequilibrio entre la intensificación y la diversificación. Como resultado, el algoritmo requiere un mayor número de iteraciones para converger a un frente de Pareto preciso.

Método de combinación de soluciones: la idea de este método es transformar un subconjunto dado de soluciones producidas por el método de generación de subconjuntos en una o más soluciones combinadas que den lugar a nuevos individuos. El método de combinación en el esquema de búsqueda dispersa es análogo al operador de cruce en algoritmos genéticos; en el algoritmo AbYSS el operador de cruce utilizado es el SBX (simulated binary crossover operator).

2.3.2 Archivo

El objetivo principal del archivo externo consiste en almacenar los individuos no dominados encontrados durante el proceso de búsqueda, a fin de mantener aquellos individuos que producen un frente de Pareto bien distribuido. La cuestión clave en la gestión del archivo consiste en decidir si una nueva solución debe ser añadida o no.

Cuando se encuentra una nueva solución en los métodos de mejora o de combinación de soluciones, se compara con las contenidas en el archivo. Si esta nueva solución está dominada por cualquier individuo del archivo (es decir, la solución está dominada por el archivo), dicha solución es descartada, de lo contrario, la solución se almacena. Si hay soluciones en el archivo que están dominadas por el nuevo elemento, se eliminan. Si el archivo alcanza su capacidad máxima permitida después de la adición de una nueva solución, se tiene que decidir qué individuo eliminar.

La estrategia utilizada cuando el archivo está lleno en otros algoritmos evolutivos basados en archivos como PAES [14], consiste en dividir el espacio de la función objetivo en una rejilla. El algoritmo AbYSS sin embargo, propone un enfoque que consiste en utilizar la *crowding-distance* del algoritmo NSGA-II [5].

2.3.3 Algoritmo AbYSS

Propuestos los cinco métodos de la búsqueda dispersa y un procedimiento para gestionar el archivo, se puede dar una visión global del algoritmo AbYSS, tal como se muestra en el algoritmo 10.

Inicialmente, se invoca el método de generación de diversificación para generar un conjunto de soluciones iniciales y a cada una de ellas se le aplica el método de mejora (línea 1). El resultado es el conjunto inicial P. Entonces, en cada iteración del bucle que se realiza a continuación, se construye el conjunto de referencia, se

Algoritmo 10 - Esquema del algoritmo AbYSS

```
1  Construir el conjunto inicial P
2  while (condición de parada no satisfecha) do
3      ActualizacionConjuntoReferencia(build=TRUE)
4      GeneracionSubconjuntos()
5      while (se generen nuevos subconjuntos) do
6          CombinacionSoluciones()
7          for each individuo combinado
8              Mejora()
9              ActualizacionConjuntoReferencia(build=FALSE)
10         end for
11         GeneracionSubconjuntos()
12     end while
13     Agregar RefSet_1 a P
14     Mover los mejores v individuos del archivo a P
15     Rellenar P con nuevas soluciones
16 end while
17 Devolver el archivo
```

invoca el método de generación de subconjuntos y el bucle principal se ejecuta hasta que el método de generación de subconjuntos deja de producir nuevos subconjuntos de soluciones (líneas 3 a 11). Entonces, hay una fase de reinicio (líneas 13 a 15), que consiste en insertar los individuos de $RefSet_1$ en P, en segundo lugar, en mover también a P los mejores v individuos del archivo de acuerdo con la *crowding-distance* y, por último, los métodos de generación de diversificación y de mejora se utilizan para producir nuevas soluciones hasta llenar P.

La idea de mover v individuos del archivo al conjunto inicial (línea 14) es la de intensificar la búsqueda hacia el frente de Pareto ya encontrado. El grado de intensificación puede variar en función del número de individuos elegido. AbYSS utiliza un valor de v que es el mínimo del tamaño del archivo y la mitad del tamaño de P. La condición de parada del algoritmo puede ser fija o puede depender de otras condiciones (suele utilizarse un número predefinido de evaluaciones).

3

Un nuevo modelo para la localización de centros semi-repulsivos

Se considera la ubicación dentro de una región geográfica determinada de una instalación o centro *semi-repulsivo*, entendiéndose como tal una instalación que puede resultar atractiva para un determinado grupo de individuos pero que, por el contrario, es no deseada por otro colectivo diferente. Supondremos que el centro a localizar no es nocivo para la salud de las personas ni para el medio ambiente, en el sentido de que sus efectos no ponen en peligro la vida de la gente, al menos directamente o de manera repentina, a diferencia de otras instalaciones como las plantas nucleares o plantas químicas que sí lo pueden hacer si ocurre un accidente. Ejemplos de centros semi-repulsivos podrían ser centrales eléctricas de gasoil o carbón, una planta de tratamiento de aguas residuales, un lugar para el entierro de residuos, un aeropuerto o una instalación militar, entre otros.

La elección de la ubicación de uno de estos centros pone en conflicto los intereses de dos grupos bien diferenciados. Por un lado, existirá una serie de sujetos para los que la proximidad de las instalaciones resulte beneficiosa, por ejemplo, en cuanto a costes de transporte se refiere, mientras que, por otro lado, estarán aquellos individuos a los que la proximidad de dichas instalaciones les incomode. Piénsese, por

ejemplo, en la aversión que pueden sentir los habitantes de una ciudad a tener uno de estos centros cerca de sus hogares.

Además, el modelo también tiene en cuenta las preocupaciones ambientales que hacen a algunas zonas no aptas para la ubicación del centro, como por ejemplo las reservas naturales de agua o embalses.

A continuación, pasamos a modelar cada uno de los objetivos enfrentados que acabamos de comentar para, posteriormente, presentar el modelo para la localización de centros semi-repulsivos desarrollado en este trabajo.

3.1 Minimización de costes de transporte

Hallar el punto del plano tal que la suma de las distancias a s puntos fijos sea mínima es el famoso problema de Weber [24] o minisum, que busca la localización de un centro de distribución para minimizar el coste total de transporte a los puntos de demanda o destino (asumiéndolo proporcional a la distancia recorrida).

Este problema fue estudiado por primera vez por el economista alemán Alfred Weber considerando dos materias primas y un mercado en el plano. Para establecer la localización óptima, Weber se basó en el criterio de minimización de costes de transporte, determinando la ubicación final de la industria como resultado del deseo de los empresarios de situarse a la menor distancia posible de las fuentes de materia prima, del resto de factores productivos (capital y trabajo) y del mercado de consumo de sus productos. De esa manera, definió la localización óptima como el lugar donde una empresa puede producir con el mínimo coste.

El problema de Weber, con todas sus variantes y extensiones, es uno de los problemas más estudiados en teoría de localización y se suele aplicar a localización de escuelas, hospitales, almacenes, etc.

Suponiendo que el centro se ubica en el punto $x = (x_1, x_2)$ del plano, el problema puede formularse como sigue:

$$\text{minimizar } \{F(x) = \sum_{i=1}^{s} \omega_i d(a_i, x)\} \tag{3.1}$$

donde a_i, $i = 1, ..., s$, es el i-ésimo lugar donde se encuentran los usuarios o i-ésimo punto de demanda y $d(a_i, x)$ es una medida de la distancia entre a_i y x. En cuanto a los parámetros ω_i, éstos son pesos que, dependiendo del problema, pueden

representar, por ejemplo, el número total de usuarios en a_i o el coste de transporte por unidad de distancia al lugar a_i.

3.2 Minimización de la repulsión

En este modelo se considera la ubicación, dentro de una región geográfica determinada, de un centro no deseado pero que no es nocivo para la salud. El modelo matemático presentado en [9] y que seguiremos aquí, minimiza la repulsión global de los habitantes de la región, teniendo en cuenta además que pueden existir zonas no aptas para la ubicación del centro por razones medioambientales.

Para modelar la repulsión de los habitantes de la ciudad a_i cuando la instalación es ubicada en el punto $x = (x_1, x_2)$ del plano, se propone usar la función:

$$rp(a_i, x) = \frac{1}{1 + exp(\alpha_i + \beta_i d(a_i, x))}, \beta_i > 0, \tag{3.2}$$

donde $d(a_i, x)$ es una medida de la distancia entre a_i y x, y α_i y β_i son dos parámetros a ser estimados para cada ciudad a_i.

Esta función permite modelar muchas intensidades diferentes de repulsión, dependiendo del tipo de instalación a localizar y de los sentimientos de los habitantes de la ciudad a_i. Cuanto menor sea el valor de α_i, mayor es la repulsión de los habitantes a la ubicación de la instalación cerca de su ciudad o en sus zonas periféricas, y cuanto mayor sea el valor de β_i, más rápido es el cambio de opinión a la hora de considerar una distancia aceptable o no (ver figura 3.1). Obsérvese que cuando la instalación se encuentra lo suficientemente lejos de una ciudad la repulsión disminuye hasta cero.

La estimación de los parámetros α_i y β_i se podría realizar, por ejemplo, llevando a cabo una encuesta pública en la ciudad a_i.

El objetivo del modelo es minimizar la repulsión global de los habitantes de las ciudades a_i, $i = 1, ..., s$, a la ubicación de la nueva instalación cuando ésta se localiza en x, es decir, minimizar la función

$$GRP(x) = \sum_{i=1}^{s} \omega_i rp(a_i, x), \tag{3.3}$$

donde ω_i es un peso positivo proporcional a la importancia de la ciudad a_i, por lo general relacionado con su número de habitantes.

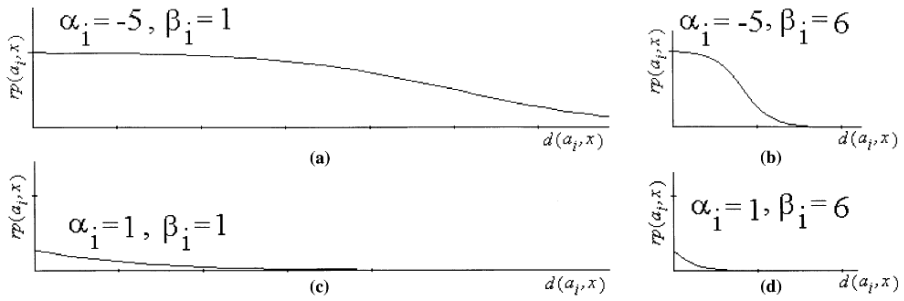

Figura 3.1: Forma de $rp(a_i, x)$ dependiendo del valor de los parámetros

No obstante, alrededor de cada ciudad habrá un área en la que se prohíbe la ubicación de las instalaciones, para evitar la posibilidad de que se ubique en una de las ciudades o demasiado cerca de ellas. Además, según el país y el tipo de instalación a ubicar, pueden existir leyes y regulaciones para prevenir que esto suceda. En el modelo presentado en [9] estas zonas prohibidas puede tener cualquier forma y si dos o más ciudades están lo suficientemente cerca pueden compartir la misma zona prohibida. El único requisito del modelo es ser capaz de especificar el exterior de las zonas prohibidas a través de un conjunto de restricciones, $g_j(x) \leq 0$, $j = 1, ..., r$.

La región geográfica donde se pretende ubicar la instalación no deseada también puede contener áreas protegidas en las que la ubicación no está permitida, debido a su interés ecológico, como parques naturales o fuentes de agua u otras áreas donde la ubicación no es posible, tales como los lagos. Estas áreas se tienen en cuenta en el modelo como áreas prohibidas. De nuevo, pueden tener cualquier forma y sólo se requiere que se pueda especificar el exterior de estas zonas mediante un conjunto de restricciones, $g_j(x) \leq 0$, $j = r + 1, ..., r'$.

La región geográfica G en la que el centro no deseado tiene que ser ubicado también puede tener cualquier forma, el único requisito es ser capaz de especificarla a través de un conjunto de restricciones, $g_j(x) \leq 0$, $j = r' + 1, ..., r''$. Por lo tanto, la región factible S puede ser definida como el conjunto $S = G - C - A$, donde C es la unión de las zonas prohibidas alrededor de las ciudades y A es la unión de las áreas protegidas y las zonas donde la localización no es posible. La región S viene dada por todas las restricciones, $g_j(x) \leq 0$, $j = 1, ..., r''$ (se supone que $S \neq \emptyset$).

3.3 Formulación del modelo

El modelo de minimización de la repulsión presentado en el apartado anterior no es adecuado para la ubicación de instalaciones semi-repulsivas como las que estamos interesados en estudiar si se utiliza $GRP(x)$ como única función objetivo. Obsérvese que $lim_{||x||\to+\infty} GRP(x) = 0$, así que la ubicación óptima con respecto a ese objetivo sería en el infinito. Sin embargo, el modelo puede ser útil para la ubicación de un centro semi-repulsivo si éste está incorporado en un modelo multicriterio, en el que $GRP(x)$ modelaría la repulsión de los habitantes a la ubicación de las nuevas instalaciones.

El modelo propuesto para la localización de un centro semi-repulsivo consiste en un modelo de optimización biobjetivo que incluye, como primer objetivo, uno de minimización de costes de transporte como el problema de Weber o minisum presentado anteriormente y, como segundo objetivo uno de minimización de repulsión. Mientras que la repulsión empujará la ubicación del centro no demasiado cerca de núcleos existentes, su atractivo, modelado por el primer objetivo, hará que la ubicación óptima tampoco esté demasiado lejos.

De este modo, la formulación del modelo quedaría:

$$\begin{array}{ll} \text{minimizar} & \{f_1(x), f_2(x)\} \\ \text{sujeto a} & x \in S \end{array} \tag{3.4}$$

donde $f_1(x) = \sum_{i=1}^{s} \omega_i d(a_i, x)$ y $f_2(x) = \sum_{i=1}^{s} \omega_i rp(a_i, x)$. Los pesos ω_i serán positivos y proporcionales a la importancia de cada una de las ciudades a_i, para $i = 1, ..., s$, relacionados con el número de habitantes. La región factible S será la definida anteriormente en la sección 3.2.

Para elegir una función de distancia se debe tener en cuenta que lo que se quiere es medir la atracción o rechazo de los habitantes a la instalación tal y como ellos lo sienten. De este modo, el modelo debe utilizar una función distancia que mida las distancias como la gente las percibe. Si fuéramos a ubicar la instalación semi-repulsiva dentro de una ciudad con una rejilla rectangular de calles, deberíamos utilizar la norma rectangular, pero si la ubicación se fuera a realizar en una región de un océano, la norma euclídea es la que debería ser seleccionada. En general, cuando la instalación se va a situar dentro de una región geográfica que contiene ciudades, en la que, por lo general, la gente sabe cuáles son las distancias entre algunos pares de ciudades, ya sea por su propia experiencia o por obtener esa información en letreros u hojas de ruta, la mejor opción es utilizar una función norma que proporcione las

distancias reales entre ciudades, que por lo general se pueden obtener de las matrices de distancia incluidas en los mapas. La función de predicción de la distancia más popular es la norma ℓ_p ponderada (véase [1]), aunque está demostrado que la norma ℓ_{2b} dada por

$$\ell_{2b}(x) = [b_1(x_1)^2 + b_2(x_2)^2]^{1/2}, b_1, b_2 > 0, \tag{3.5}$$

es competitiva con la anterior [10].

4

Experiencia computacional

4.1 jMetal

jMetal [7, 6] es sinónimo de **al**goritmos **Met**aheurísticos en **j**ava, y es una plataforma orientada a objetos basada en java dirigida al desarrollo, experimentación y estudio de metaheurísticas para la resolución de problemas de optimización multiobjetivo. Se encuentra bajo la Licencia Pública General de GNU y se puede obtener libremente de `http://jmetal.sourceforge.net`.

jMetal proporciona un rico conjunto de clases que pueden ser utilizadas como bloques de construcción de metaheurísticas multiobjetivo; haciendo uso de la reutilización de código, los algoritmos comparten los mismos componentes básicos, tales como las implementaciones de los operadores genéticos y los estimadores de densidad, lo que facilita no sólo el desarrollo de nuevas técnicas multiobjetivo, sino también el llevar a cabo diferentes tipos de estudios.

En la plataforma están implementadas un gran número de las más avanzadas técnicas metaheurísticas multiobjetivo y también están disponibles muchos de los problemas de prueba incluidos en la mayoría de estudios actuales. jMetal también proporciona indicadores de calidad para evaluar el rendimiento, así como un conjunto de utilidades que ayudan en la realización de estudios experimentales.

A continuación se utiliza el Lenguaje Unificado de Modelado (LUM o UML, por

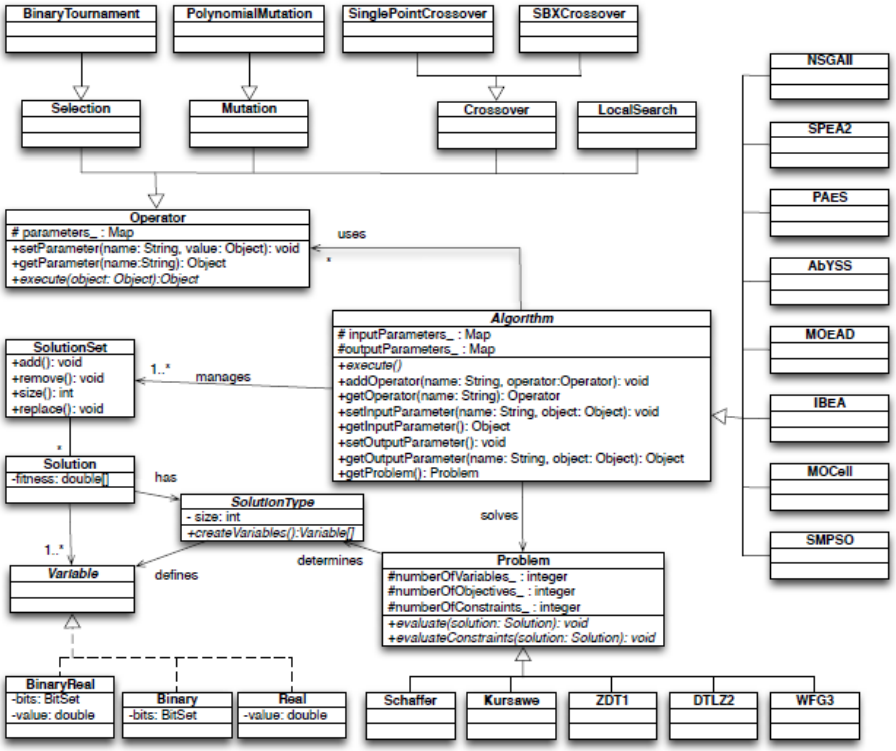

Figura 4.1: jMetal. Diagrama de clases.

sus siglas en inglés, Unified Modelling Language) para describir la arquitectura y los componentes de jMetal. En la figura 4.1 se muestra un diagrama de clases UML que representa los componentes principales y sus relaciones. El diagrama es una versión simplificada con el fin de hacerlo comprensible. La arquitectura básica de jMetal se basa en un *algoritmo* que resuelve un *problema* haciendo uso de uno o más conjuntos de soluciones (*SolutionSet*) y un conjunto de objetos *operador*. jMetal utiliza una terminología genérica para nombrar a las clases con el fin de hacerlos útiles para cualquier metaheurística. En el contexto de los algoritmos evolutivos, las poblaciones y los individuos corresponden a los objetos *SolutionSet* y *Solution* de jMetal, respectivamente.

En jMetal, todos los problemas heredan de la clase *Problem*. Esta clase contiene dos métodos básicos: *evaluate()* y *evaluateConstraints()*. Ambos métodos reciben una *Solution* que representa una solución candidata del problema, el primero la evalúa, y el segundo determina la violación general de las restricciones de esta solución. Todos los problemas tienen que definir el método *evaluate()*, mientras que sólo los problemas con restricciones necesitan definir *evaluateConstraints()*. Por defecto,

el mecanismo de control aplicado a las restricciones es el propuesto en [5]. Una característica clave del diseño de jMetal es que es en el propio problema donde se definen los tipos de soluciones permitidos para su resolución.

4.2 Algoritmos de resolución. Parámetros

Haciendo uso de la plataforma jMetal se ha estudiado el funcionamiento de los algoritmos genéticos multiobjetivo NSGA-II, SPEA2 y AbYSS a la hora de resolver problemas de localización de un centro semi-repulsivo de diferentes tamaños. El diseño de estos problemas y los resultados obtenidos son tratados en las siguientes secciones. Para poder comparar los resultados proporcionados por los tres algoritmos en estudio, éstos han sido configurados del mismo modo. Detallamos a continuación los parámetros de configuración utilizados.

En los tres algoritmos tratados, en cada ejecución de cada uno de los problemas resueltos, se realizan 25.000 evaluaciones de las funciones objetivo antes de proporcionar la mejor aproximación encontrada del frente de Pareto que, en todos los casos, tiene un tamaño de 100 puntos, es decir, $M = 100$. Otro punto clave en la configuración de los algoritmos genéticos es el tamaño de la población (N) con la que se trabaja en cada iteración. En NSGA-II y SPEA2 se he trabajado con una población de 100 individuos, mientras que en AbYSS se ha utilizado un tamaño de la población igual a 20. En cada iteración de AbYSS se trabaja con estos 20 individuos en la población y con otros 20 que son los que forman el conjunto de referencia, en concreto $RefSet_1$=$RefSet_2$=10. El tamaño del archivo en los casos de SPEA2 y AbYSS ha sido $\bar{N} = 100$.

En NSGA-II, a partir de la población de tamaño 100, se construye una nueva generación, también de tamaño 100, y de esos 200 puntos se seleccionan los 100 mejores haciendo uso del *crowded-comparison operator* que son los que forman la aproximación del frente de Pareto. En SPEA2 y AbYSS es en el archivo donde se guarda la aproximación del frente de Pareto que se proporciona como solución.

Los operadores genéticos utilizados en los algoritmos son: el cruce SBX (Simulated Binary Crossover), la mutación polinomial y la selección por torneo binario. En el caso del algoritmo AbYSS los operadores que son necesarios especificar son los de cruce y mejora, siendo en este último donde se incluye la mutación polinomial, ya que el método de mejora realiza una búsqueda local basada en ese operador de mutación. Por lo que respecta al operador de selección, en SPEA2 el criterio de selección

es por dominancia, mientras que en NSGA-II primero se compara la dominancia y, en caso de que los dos pertenezcan al mismo frente, se utiliza la *crowding-distance* para decidir, pues es así como funciona el *crowded-comparison operator*. La probabilidad de mutación utilizada ha sido 0.5 en todos los casos (1/número de variables) y la probabilidad de cruce se ha fijado en 0.9 para los algoritmos NSGA-II y SPEA2 y en 1 para AbYSS.

4.3 Indicadores de calidad

Para evaluar el funcionamiento de los algoritmos a la hora de resolver diferentes problemas se suelen considerar dos aspectos; por un lado, la distancia entre el frente de Pareto proporcionado por el algoritmo en cuestión y el frente de Pareto real del problema debería ser lo menor posible y, por otro lado, la dispersión de las soluciones encontradas debería ser lo mayor posible para obtener una distribución suave y uniforme.

Para determinar estas características es necesario conocer la localización exacta del frente de Pareto real. En nuestro caso, puesto que desconocemos este frente real, lo hemos aproximado a partir de los resultados obtenidos en las ejecuciones de los problemas en los tres algoritmos. Como sabemos, cada algoritmo en cada ejecución proporciona una aproximación al frente de Pareto de 100 puntos y, como comentaremos en el siguiente apartado, cada problema ha sido ejecutado 100 veces con cada algoritmo. Por tanto, disponemos de $100 \cdot 3 \cdot 100 = 30.000$ puntos en cada problema para calcular el frente "óptimo" de Pareto. Para cada problema, se han recorrido esos 30.000 puntos seleccionando únicamente aquellos puntos no-dominados y se ha considerado a dicho conjunto de puntos como la mejor aproximación posible al frente de Pareto real.

Para obtener resultados fiables, antes de calcular cualquier medida, se normalizan los valores objetivo de cada una de las funciones objetivo del problema, según la siguiente fórmula:

$$z_i\_normalizado = (z_i - z_{i,\min})/(z_{i,\max} - z_{i,\min})$$

donde $z_{i,\max}$ y $z_{i,\min}$ son los valores máximo y mínimo de la i-ésima función objetivo en el frente de Pareto óptimo.

Los indicadores de calidad pueden ser clasificados en tres categorías, dependiendo de si éstos evalúan la proximidad al frente de Pareto, la diversidad en las soluciones obtenidas o ambas. En este estudio se ha utilizado una medida de cada tipo.

Distancia Generacional: fue introducida por Van Veldhuizen y Lamont [23] para medir la distancia entre el conjunto ofrecido como aproximación al frente de Pareto y el frente de Pareto óptimo. La distancia generacional se define como sigue:

$$GD = \frac{\sqrt{\sum_{i=1}^{M} D_i^2}}{M} \tag{4.1}$$

donde M es el número de vectores en el conjunto ofrecido como aproximación del frente de Pareto y D_i es la distancia euclídea (medida es el espacio objetivo) entre cada uno de estos vectores y el miembro más cercano del frente de Pareto óptimo. Un valor de $GD = 0$ nos indicaría que todos los elementos generados se encuentran en el frente de Pareto óptimo.

Spread: este indicador [5] mide el grado de dispersión entre las soluciones obtenidas, interesando, en este sentido, obtener un conjunto de soluciones que se extienda por todo el frente de Pareto óptimo de la forma más uniforme posible. Se calcula la distancia euclídea d_i entre soluciones consecutivas en el conjunto ofrecido como aproximación del frente de Pareto y el promedio $\bar{d}$ de todas estas distancias d_i, $i = 1, 2, ..., (M-1)$, (con M soluciones hay $(M-1)$ distancias consecutivas). Se calculan también las distancias entre las soluciones extremas (en el espacio objetivo) y los correspondientes extremos del frente de Pareto óptimo, que denotaremos por d_f y d_l. Haciendo uso de la siguiente métrica se calcula entonces la falta de uniformidad en la distribución (véase la figura 4.2):

$$\Delta = \frac{d_f + d_l + \sum_{i=1}^{M-1} |d_i - \bar{d}|}{d_f + d_l + (M-1)\bar{d}}. \tag{4.2}$$

Una buena distribución haría que todas las distancias d_i fueran iguales a $\bar{d}$ y haría que $d_f = d_l = 0$ con lo que, en el conjunto de soluciones no dominadas más ampliamente y uniformemente distribuido, el numerador de Δ sería cero, haciendo que este indicador tome valor cero (para cualquier otra distribución, el valor de Δ sería mayor que cero). Para dos distribuciones con idénticos valores de d_f y d_l, Δ tendrá un valor mayor con una peor distribución de las

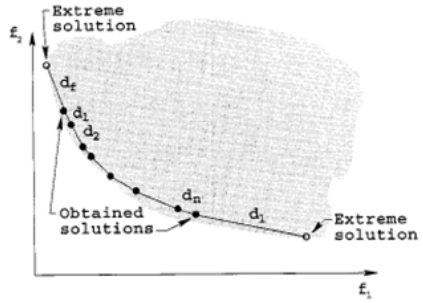

Figura 4.2: Medida de diversidad Δ

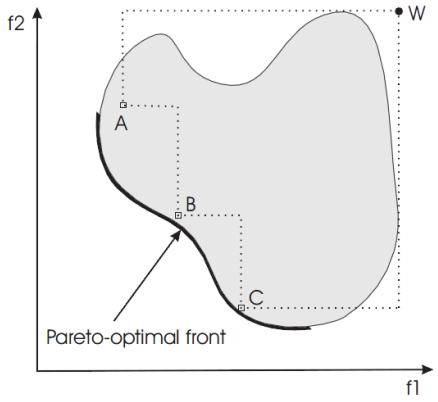

Figura 4.3: Hipervolumen delimitado por soluciones no-dominadas

soluciones no extremas. Nótese que, en un escenario en el que haya una gran variación en d_i, el máximo de este indicador podría ser mayor que uno.

Hipervolumen: se llama así al volumen (en el espacio objetivo) cubierto por los puntos que forman la aproximación del frente de Pareto F [25] (región marcada por las lineas discontinuas en la figura 4.3, $F = \{A, B, C\}$).

Matemáticamente, para cada solución $j \in F$, se construye un hipercubo v_j cuyas esquinas opuestas en la diagonal principal son la propia solución j y un *punto de referencia* W. Como punto de referencia se toma un vector cuyas componentes son los peores valores de las funciones objetivo. El hipervolumen puede ser entonces calculado como el volumen de la unión de todos los

hipercubos:

$$HV = volumen(\bigcup_{j=1}^{|F|} v_j) \tag{4.3}$$

Obviamente, son preferibles los algoritmos que proporcionan mayores valores de HV.

4.4 Problemas test

Para definir un problema en jMetal debemos indicar el número de variables, de funciones objetivo y de restricciones que lo forman, así como implementar los métodos *evaluate()* (en todos los casos) y *evaluateConstraints()* (sólo para los problemas que tengan restricciones para poder obtener la violación general de las mismas). En nuestro caso, debemos implementar ambos métodos puesto que nos encontramos ante problemas con restricciones. En concreto, tendremos dos variables, (x_1, x_2), que nos informarán de la ubicación en el plano del centro semi-repulsivo, trabajaremos con dos funciones objetivo (la primera de ellas referente a los costes de transporte y la segunda a la repulsión) y el número de restricciones dependerá del problema concreto que estemos tratando.

Recordemos que nuestro problema es el siguiente:

$$\begin{array}{ll} \text{minimizar} & \{\sum_{i=1}^{s} \omega_i \ell_{2b}(a_i, x),\ \sum_{i=1}^{s} \omega_i rp(a_i, x)\} \\ \text{sujeto a} & g_j(x) \leq 0,\ \forall j = 1, ..., r'' \end{array} \tag{4.4}$$

donde la función de repulsión viene dada por

$$rp(a_i, x) = \frac{1}{1 + exp(\alpha_i + \beta_i \ell_{2b}(a_i, x))}, \beta_i > 0 \tag{4.5}$$

Los algoritmos evolutivos multiobjetivo en estudio se han probado en una amplia variedad de problemas. Como se ha comentado, se ha utilizado la norma ℓ_{2b} como función distancia (aunque se podría haber escogido cualquier otra), generando aleatoriamente los parámetros del modelo a partir de distribuciones uniformes en los siguiente intervalos: $b_1, b_2 \in [1, 2.5]$, $\alpha_i \in [-5, 1]$ y $\beta_i \in [1, 6]$. Los pesos ω_i se encuentran en el intervalo $[0, 10]$ y han sido asignados de forma proporcional al número de habitantes de cada ciudad (número aleatorio entre 10.000 y 100.000).

El área sobre la que trabajamos es el cuadrado $([0, 10], [0, 10])$ y las coordenadas de las ciudades ubicadas en él se han generado también a partir de una distribución uniforme donde cada ciudad tiene a su alrededor un área prohibida de tamaño proporcional a su peso, concretamente, un círculo con centro en a_i y radio ω_i.

Las áreas protegidas también son circunferencias generadas aleatoriamente en el cuadrado $([0, 10], [0, 10])$, siendo su número una cantidad aleatoria comprendida entre el 5% y el 20% del número de ciudades. El radio de las áreas protegidas se ha calculado como sigue: radio = (número aleatorio entre 5 y 20)/(número ciudades).

Con todo esto, las restricciones necesarias para definir cada uno de los problemas serán: 4 restricciones lineales para especificar el área geográfica en estudio, es decir, el cuadrado $([0, 10], [0, 10])$, más un círculo por cada una de las ciudades, más tantos círculos como áreas protegidas conste el problema (al estar ambas definidas por círculos, cada una de ellas puede especificarse por medio de una sola restricción).

El número de ciudades consideradas ha sido $s = 10, 25, 50, 100, 200, 500$, generándose 10 problemas de cada tamaño. Por tanto, se han tratado un total de 60 problemas, realizándose 100 ejecuciones de cada algoritmo sobre cada uno de ellos.

4.5 Generación, ejecución y representación gráfica de los problemas

A través de la plataforma jMetal y el lenguaje de programación java, la generación y ejecución de los problemas test que acabamos de describir se ha realizado como se indica a continuación.

Se ha implementado el fichero *GenerateProblem.java* para generar todos los problemas posteriormente analizados en este trabajo. En la clase java *GenerateProblem* se ha implementado el método *generateFile* que crea un problema de localización de un centro semi-repulsivo como los que acabamos de describir, indicándole únicamente el número de ciudades que queremos incluir y el nombre que queremos darle al problema. De este modo, con un simple bucle, somos capaces de generar los 60 problemas con los que hemos trabajado. Como ya indicamos, en jMetal es en el propio problema donde se definen los tipos de soluciones permitidos para su resolución, por lo que el método *generateFile*, al crear cada uno de los problemas, también debe indicar que, en nuestro caso, las variables deben ser de tipo "Real".

Para evaluar el funcionamiento de los algoritmos, se ejecutarán sobre los pro-

blemas test descritos y se analizarán los indicadores de calidad mencionados. Todo este proceso puede realizarse de un modo relativamente sencillo a través del paquete *jmetal.experiments*.

Las 100 ejecuciones independientes de los 60 problemas generados se realizan en la clase java *Location* implementada para tal fin y que hereda de la clase *Experiment* de jMetal. Suponiendo que cada algoritmo ha sido configurado en su clase de ajuste correspondiente, la implementación de la clase *Location* es sencilla. Debemos indicar al "experimento" una lista con los algoritmos a utilizar, otra lista con los problemas a resolver y el número de ejecuciones que queremos llevar a cabo de cada uno de ellos.

Generados los problemas y realizadas todas las ejecuciones independientes, el siguiente paso es analizar los resultados. Como se ha dicho anteriormente, para poder calcular los indicadores de calidad es necesario conocer el frente de Pareto óptimo. En nuestro caso, puesto que desconocemos este frente óptimo, lo hemos aproximado a partir de los resultados obtenidos en las ejecuciones de los tres algoritmos. Esta aproximación se realiza en el fichero *GenerateParetoFront.java*. En esta clase java se ha implementado el método *generateFront* al que hay que indicarle el problema del que vamos a calcular su frente de Pareto "óptimo" y el número de ejecuciones (ficheros de salida de los algoritmos NSGA-II, SPEA2 y AbYSS) que tiene que analizar.

Cuando cualquiera de los algoritmos proporcionados por jMetal se ejecuta para resolver un problema, se devuelven dos ficheros que contienen las aproximaciones de las soluciones óptimas (en el espacio de decisión) y del frente de Pareto (en el espacio imagen). Por defecto, estos ficheros se llaman VAR y FUN, respectivamente. Estos ficheros FUN son los que se utilizan para calcular los indicadores de calidad. Entonces, obtenidas las aproximaciones de los frentes de Pareto reales y haciendo uso del paquete *jmetal.qualityIndicator* (destinado a facilitar el uso de indicadores de calidad dentro de los programas), se ha desarrollado el fichero *LocationQuality.java* que calcula la distancia generacional, el spread y el hipervolumen para todos los problemas estudiados.

Para que sea posible la visualización de los resultados de los problemas de localización de un centro semi-repulsivo que se han estudiado, se ha utilizado la herramienta Tcl-Tk (originado del acrónimo en inglés Tool command language - Tool kit) que es un lenguaje de script utilizado para la creación de interfaces gráficas. El método *generateFile* de la clase java *GenerateProblem* que hemos comentado anteriormente, además de crear el propio problema de localización a resolver, también

genera el fichero necesario para dibujar cada problema.

En el fichero C++ *genera_fichero.cpp* se ha implementado un programa que sirve para generar los ficheros de lectura de la herramienta Tcl-Tk. El fichero ejecutable *genera_fichero* se utiliza como sigue:

genera_fichero datos VAR FUN COLOR espacio_origen frente_pareto

donde:

- *datos* es el fichero creado por el método *generateFile* de la clase java *GenerateProblem* y contiene las cotas inferior y superior de las variables x_1 y x_2 y el número de puntos de demanda y áreas protegidas, con sus localizaciones y radios, del problema a representar.

- *VAR* y *FUN* son los ficheros proporcionados por jMetal al resolver dicho problema y serán utilizados para dibujar el conjunto de puntos eficientes en el espacio origen dado en *datos* y su correspondiente aproximación del frente de Pareto. Los ficheros VAR y FUN no se utilizan en el programa *genera_fichero* tal cual los proporciona jMetal, si no que los valores de cada una de las dos funciones objetivo del fichero FUN son normalizados, dividiéndolos entre el valor máximo de su función correspondiente para obtener una representación gráfica de ambas funciones en el intervalo [0,1]. Además, los ficheros son ordenados de modo ascendente según el valor de la primera función objetivo (costes de transporte), lo que será útil para la escala de colores utilizada en la representación gráfica.

- *COLOR* es un fichero que contiene la codificación en hexadecimal de cien tonalidades diferentes de gris. En la próxima sección se explica detalladamente el uso de esta escala de grises.

- *espacio_origen* y *frente_pareto* son los nombres que tendrán los ficheros que se generan con la codificación que debe usarse en la herramienta Tcl-Tk para representar gráficamente el espacio origen con el conjunto de puntos eficientes y el frente de Pareto, respectivamente.

Generados los dos ficheros anteriores y mediante el uso del fichero *escala.tcl* se obtienen los gráficos deseados del siguiente modo:

escala.tcl espacio_origen
escala.tcl frente_pareto

4.6 Resultados

A continuación mostramos los resultados obtenidos con los algoritmos NSGA-II, SPEA2 y AbYSS. Como se ha indicado, con cada algoritmo se han realizado 100 ejecuciones independientes de cada uno de los 60 problemas generados, a los que se ha nombrado LPXX_YY, donde $XX = \{01, 02, ..., 10\}$ representa el número de problema e $YY = \{10, 25, 50, 100, 200, 500\}$ el número de ciudades de dicho problema.

En las figuras 4.4, 4.5 y 4.6 puede observarse la aproximación del conjunto de puntos eficientes y la correspondiente aproximación del frente de Pareto proporcionadas por los tres algoritmos en estudio para el problema LP01_25 (este problema en concreto tiene 4 áreas protegidas). Un ejemplo de mayor tamaño puede verse en las figuras 4.7, 4.8 y 4.9, donde se representa la solución proporcionada por los tres algoritmos para el problema LP01_100, con 19 áreas protegidas.

En dichas figuras las regiones prohibidas en torno a las ciudades se muestran en color amarillo y las áreas protegidas en verde. Además, como se ha comentado, las funciones f_1 (costes de transporte) y f_2 (repulsión) han sido normalizadas en el rango $[0, 1]$ antes de representarlas. El frente de Pareto se ha dibujado en una escala de grises de forma que, el color más oscuro representa el punto donde los costes de transporte hasta la ubicación del centro semi-repulsivo se minimizan y la función de repulsión toma su peor valor (el más alto). Tal y como puede apreciarse en los gráficos, el gris va paulatinamente disminuyendo de intensidad lo que representa que los costes de transporte aumentan mientras que la repulsión global de los habitantes de las ciudades disminuye, hasta llegar al punto más claro donde se minimiza la repulsión y los costes de transporte son máximos. En esta misma escala de grises se han representado sobre el espacio origen las diferentes localizaciones del centro semi-repulsivo, para poder visualizar con facilidad qué ubicaciones se corresponden con qué puntos del frente de Pareto. Esta representación gráfica ha sido posible gracias al uso de la herramienta Tcl-Tk.

Tanto en el ejemplo de 25 ciudades como en el de 100, pueden apreciarse las discontinuidades de los frentes de Pareto debidas a la imposibilidad de ubicar el centro en ciertas áreas de la región geográfica (áreas protegidas y áreas alrededor de las ciudades). A simple vista, los resultados de los algoritmos NSGA-II y AbYSS parecen mucho más similares entre sí, que los proporcionados por SPEA2, al igual que parece que este último algoritmo no proporciona una diversidad de soluciones tan buena como los otros dos algoritmos en estudio.

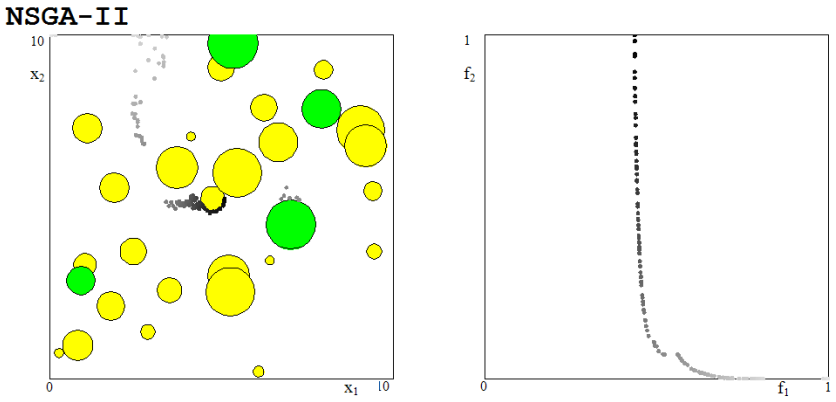

Figura 4.4: NSGA-II. Espacio origen y frente de Pareto. Problema LP01_25, con 4 áreas protegidas

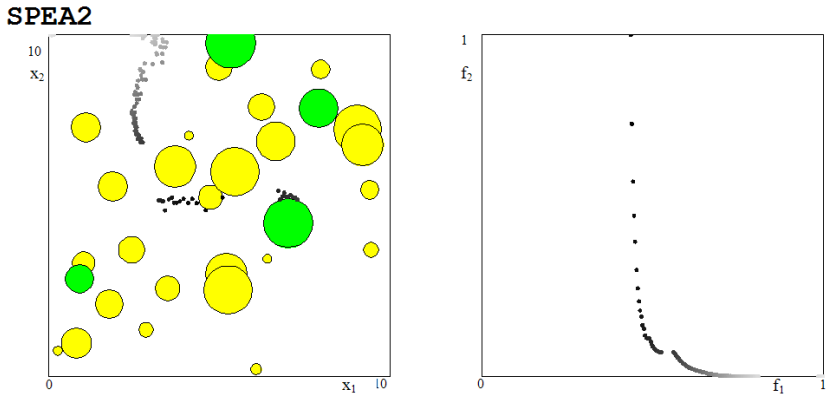

Figura 4.5: SPEA2. Espacio origen y frente de Pareto. Problema LP01_25, con 4 áreas protegidas

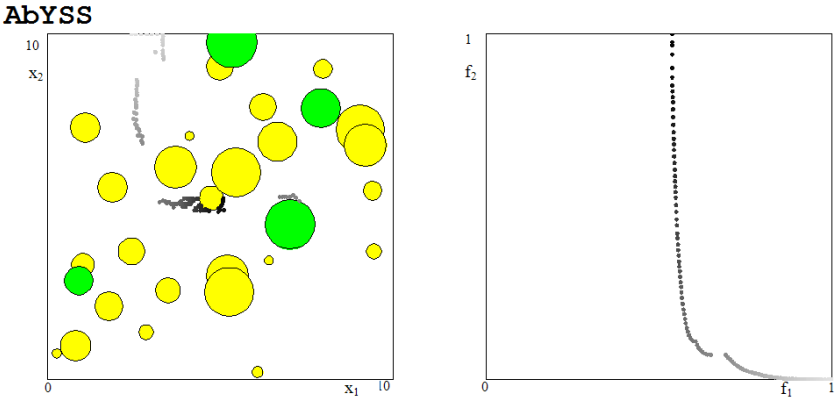

Figura 4.6: AbYSS. Espacio origen y frente de Pareto. Problema LP01_25, con 4 áreas protegidas

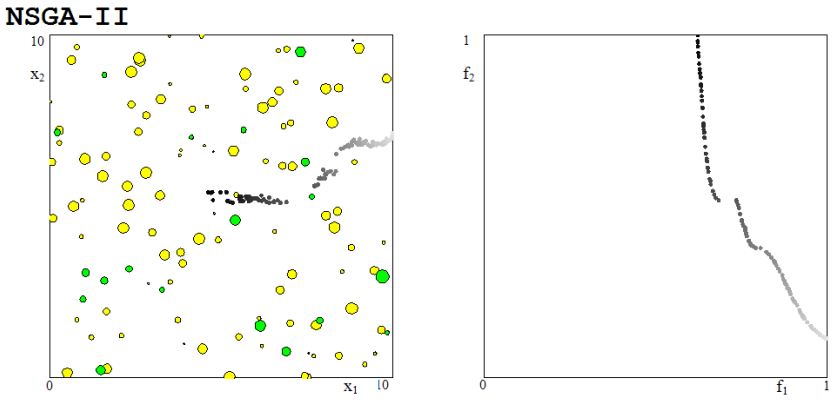

Figura 4.7: NSGA-II. Espacio origen y frente de Pareto. Problema LP01_100, con 19 áreas protegidas

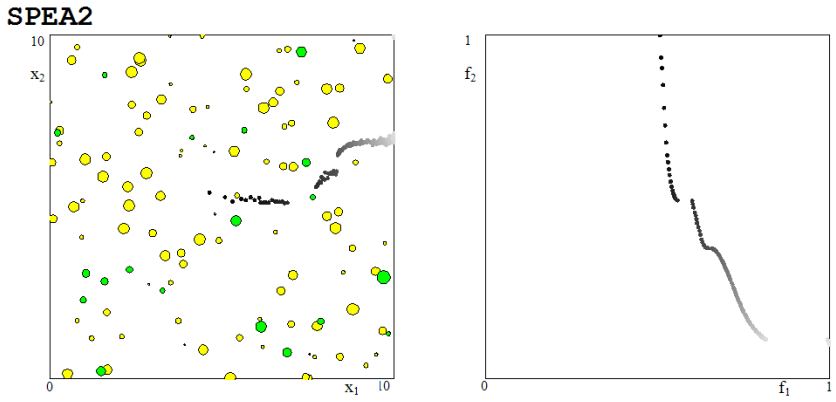

Figura 4.8: SPEA2. Espacio origen y frente de Pareto. Problema LP01_100, con 19 áreas protegidas

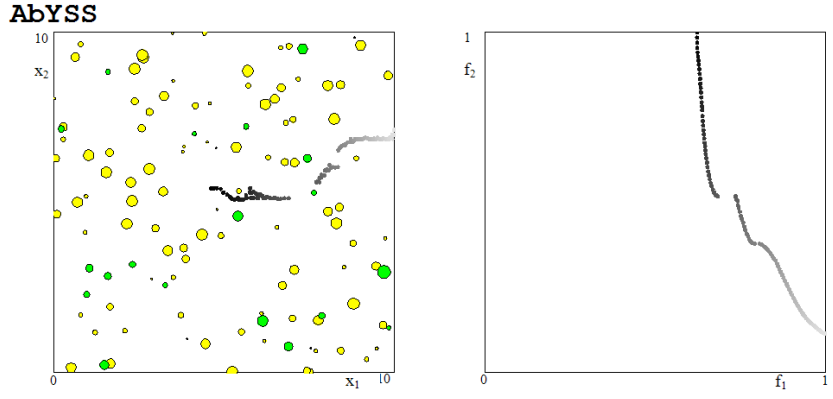

Figura 4.9: AbYSS. Espacio origen y frente de Pareto. Problema LP01_100, con 19 áreas protegidas

Para analizar el comportamiento de los tres AEMOs de un modo más exhaustivo, se presenta a continuación un resumen de los tiempos de ejecución y de los tres indicadores de calidad calculados (distancia generacional, spread e hipervolumen) para los 60 problemas de localización tratados.

Para cada problema, tanto para el tiempo de ejecución como para los indicadores de calidad, se muestra la media y la desviación estándar de sus 100 ejecuciones. En todas las tablas resumen se han agrupado los resultados por número de ciudades, se ha calculado la media para cada grupo y se ha añadido la media global de todos los problemas resueltos por cada algoritmo. Para que la interpretación de los resultados resulte más cómoda, los datos mostrados para cada problema se han coloreado de la siguiente forma: en gris oscuro se representa el mejor resultado, en gris menos intenso se indica el segundo mejor algoritmo en cuanto a calidad del indicador considerado y en blanco aparece el algoritmo con peor comportamiento para ese problema.

En la tabla 4.1 se indican las medias y las desviaciones estándar de los tiempos de ejecución, en segundos, para los tres algoritmos. Las ejecuciones de todos los problemas se han llevado a cabo en un ordenador portátil LG E500, con sistema operativo Windows XP, procesador Intel (R) Core (TM) Duo CPU T7250 @ 2.00GHz 1.18GHz y 2.00GB de RAM.

Se observa claramente como los tiempos de ejecución del algoritmo AbYSS son, en general, mejores que los de los otros dos algoritmos. El algoritmo NSGA-II necesita más tiempo que AbYSS pero consigue incluso mejorarlo en algunos casos concretos (LP09_10, LP10_25, LP09_50, LP10_50, LP09_100 y LP10_100). Sin embargo, el algoritmo SPEA2 tiene unos tiempos de ejecución muy superiores (la media global de SPEA2 se sitúa en 22 segundos mientras que las de NSGA-II y AbYSS son de 5.82 segundos y 4.76 segundos, respectivamente).

Tabla 4.1: Tiempos ejecución (segundos). Media y desviación estándar

	NSGA-II	SPEA2	AbYSS
LP01_10	$2.29e+00_{2.21e-02}$	$1.23e+01_{1.22e-01}$	$1.20e+00_{6.48e-02}$
LP02_10	$2.98e+00_{3.97e-02}$	$2.52e+01_{2.13e-01}$	$2.35e+00_{9.36e-02}$
LP03_10	$3.05e+00_{2.47e-02}$	$1.63e+01_{1.18e-01}$	$1.60e+00_{4.34e-02}$
LP04_10	$3.01e+00_{1.99e-02}$	$2.68e+01_{2.01e-01}$	$2.15e+00_{8.59e-02}$
LP05_10	$2.89e+00_{2.19e-02}$	$1.97e+01_{5.01e-01}$	$1.99e+00_{8.96e-02}$
LP06_10	$2.74e+00_{2.60e-02}$	$1.84e+01_{1.62e-01}$	$1.78e+00_{6.75e-02}$
LP07_10	$2.76e+00_{2.54e-02}$	$1.56e+01_{1.53e-01}$	$1.52e+00_{1.21e-01}$
LP08_10	$2.72e+00_{1.31e-02}$	$1.79e+01_{3.52e-01}$	$1.76e+00_{5.16e-02}$
LP09_10	$1.73e+00_{3.65e-02}$	$2.79e+01_{3.63e-01}$	$1.77e+00_{2.04e-01}$
LP10_10	$1.79e+00_{1.85e-02}$	$2.43e+01_{1.67e+00}$	$1.72e+00_{7.10e-02}$
Media 10 ciudades	$2.60e+00_{2.48e-02}$	$2.05e+01_{3.86e-01}$	$1.79e+00_{8.92e-02}$
LP01_25	$2.52e+00_{1.43e-02}$	$1.52e+01_{1.63e-01}$	$1.39e+00_{5.83e-02}$
LP02_25	$3.29e+00_{2.51e-02}$	$1.91e+01_{1.69e-01}$	$1.96e+00_{7.75e-02}$
LP03_25	$3.32e+00_{2.68e-02}$	$1.93e+01_{2.11e-01}$	$1.85e+00_{7.93e-02}$
LP04_25	$3.33e+00_{2.16e-02}$	$1.72e+01_{1.54e-01}$	$1.84e+00_{7.42e-02}$
LP05_25	$3.32e+00_{2.23e-02}$	$1.88e+01_{1.23e-01}$	$2.10e+00_{7.76e-02}$
LP06_25	$3.00e+00_{1.09e-02}$	$2.05e+01_{1.91e-01}$	$2.04e+00_{4.70e-02}$
LP07_25	$3.02e+00_{1.26e-02}$	$1.72e+01_{1.46e-01}$	$1.82e+00_{6.24e-02}$
LP08_25	$3.00e+00_{1.96e-02}$	$1.98e+01_{1.45e-01}$	$2.04e+00_{7.27e-02}$
LP09_25	$1.98e+00_{1.47e-02}$	$2.40e+01_{2.84e-01}$	$1.92e+00_{9.60e-02}$
LP10_25	$1.95e+00_{1.42e-02}$	$2.68e+01_{2.78e-01}$	$2.15e+00_{1.13e-01}$
Media 25 ciudades	$2.87e+00_{1.82e-02}$	$1.98e+01_{1.86e-01}$	$1.91e+00_{7.58e-02}$
LP01_50	$2.92e+00_{1.49e-02}$	$1.36e+01_{6.94e-01}$	$1.45e+00_{6.13e-02}$
LP02_50	$3.69e+00_{2.90e-02}$	$1.85e+01_{1.50e-01}$	$2.19e+00_{6.69e-02}$
LP03_50	$3.74e+00_{5.68e-02}$	$1.84e+01_{4.91e-01}$	$2.45e+00_{1.63e-01}$
LP04_50	$3.75e+00_{2.64e-02}$	$1.74e+01_{1.34e-01}$	$2.41e+00_{6.97e-02}$
LP05_50	$3.75e+00_{2.44e-02}$	$1.77e+01_{4.22e-01}$	$2.18e+00_{6.24e-02}$
LP06_50	$3.46e+00_{2.23e-02}$	$1.82e+01_{1.13e+00}$	$2.08e+00_{1.71e-01}$
LP07_50	$3.47e+00_{2.12e-02}$	$1.85e+01_{3.96e-01}$	$2.00e+00_{1.08e-01}$
LP08_50	$3.47e+00_{1.26e-02}$	$1.34e+01_{1.11e-01}$	$1.87e+00_{5.78e-02}$
LP09_50	$2.27e+00_{1.97e-02}$	$2.65e+01_{3.08e-01}$	$2.63e+00_{7.60e-02}$
LP10_50	$2.71e+00_{2.99e-01}$	$3.25e+01_{3.67e+00}$	$2.85e+00_{3.35e-01}$
Media 50 ciudades	$3.32e+00_{5.26e-02}$	$1.95e+01_{7.51e-01}$	$2.21e+00_{1.17e-01}$
LP01_100	$3.99e+00_{4.64e-01}$	$1.68e+01_{1.97e+00}$	$2.42e+00_{2.97e-01}$
LP02_100	$4.51e+00_{2.85e-02}$	$1.94e+01_{1.98e+00}$	$2.76e+00_{1.16e-01}$
LP03_100	$4.51e+00_{2.87e-02}$	$1.91e+01_{2.57e-01}$	$2.92e+00_{9.01e-02}$
LP04_100	$4.54e+00_{2.87e-02}$	$1.67e+01_{1.03e-01}$	$2.85e+00_{7.60e-02}$
LP05_100	$4.53e+00_{2.99e-02}$	$1.68e+01_{1.37e-01}$	$2.77e+00_{8.22e-02}$
LP06_100	$4.14e+00_{1.38e-02}$	$1.62e+01_{1.13e-01}$	$2.62e+00_{8.02e-02}$
LP07_100	$4.13e+00_{1.15e-02}$	$2.07e+01_{1.74e-01}$	$2.95e+00_{1.25e-01}$
LP08_100	$4.11e+00_{1.41e-02}$	$1.53e+01_{1.40e-01}$	$2.66e+00_{7.02e-02}$
LP09_100	$2.85e+00_{1.55e-02}$	$2.65e+01_{2.50e-01}$	$3.16e+00_{7.05e-02}$
LP10_100	$3.68e+00_{2.46e-02}$	$3.32e+01_{6.58e-01}$	$3.74e+00_{1.45e-01}$
Media 100 ciudades	$4.10e+00_{6.59e-02}$	$2.01e+01_{5.78e-01}$	$2.88e+00_{1.15e-01}$
LP01_200	$7.77e+00_{3.79e-02}$	$2.50e+01_{1.19e+00}$	$6.42e+00_{1.23e-01}$
LP02_200	$7.75e+00_{4.58e-02}$	$1.98e+01_{3.45e-01}$	$6.28e+00_{1.16e-01}$
LP03_200	$7.94e+00_{5.02e-02}$	$2.04e+01_{1.49e-01}$	$6.75e+00_{7.21e-02}$
LP04_200	$7.84e+00_{3.21e-02}$	$1.95e+01_{1.36e-01}$	$6.32e+00_{5.68e-02}$
LP05_200	$7.90e+00_{5.43e-02}$	$2.38e+01_{2.23e-01}$	$6.65e+00_{9.06e-02}$
LP06_200	$7.24e+00_{1.38e-02}$	$1.96e+01_{1.73e-01}$	$6.16e+00_{7.29e-02}$
LP07_200	$7.29e+00_{1.91e-02}$	$1.82e+01_{1.23e-01}$	$5.97e+00_{6.91e-02}$
LP08_200	$7.20e+00_{2.50e-02}$	$1.73e+01_{4.20e-01}$	$5.77e+00_{8.10e-02}$
LP09_200	$6.07e+00_{1.92e-02}$	$3.10e+01_{1.97e+00}$	$5.51e+00_{1.54e-01}$
LP10_200	$7.69e+00_{4.09e-02}$	$2.56e+01_{3.28e+00}$	$6.68e+00_{2.48e-01}$
Media 200 ciudades	$7.47e+00_{3.38e-02}$	$2.20e+01_{8.01e-01}$	$6.25e+00_{1.08e-01}$
LP01_500	$1.51e+01_{4.89e-02}$	$3.52e+01_{3.88e-01}$	$1.44e+01_{8.73e-02}$
LP02_500	$1.51e+01_{4.91e-02}$	$2.91e+01_{1.82e-01}$	$1.42e+01_{9.44e-02}$
LP03_500	$1.54e+01_{4.97e-02}$	$2.77e+01_{2.68e-01}$	$1.46e+01_{1.30e-01}$
LP04_500	$1.51e+01_{8.39e-02}$	$2.74e+01_{1.44e-01}$	$1.42e+01_{1.02e-01}$
LP05_500	$1.53e+01_{9.10e-02}$	$2.90e+01_{1.46e+00}$	$1.41e+01_{1.36e-01}$
LP06_500	$1.39e+01_{2.62e-02}$	$2.51e+01_{2.35e-01}$	$1.30e+01_{9.25e-02}$
LP07_500	$1.40e+01_{2.02e-02}$	$2.38e+01_{5.38e-01}$	$1.30e+01_{1.18e-01}$
LP08_500	$1.38e+01_{2.30e-02}$	$2.87e+01_{2.01e-01}$	$1.29e+01_{1.46e-01}$
LP09_500	$1.23e+01_{2.64e-02}$	$3.43e+01_{5.95e-01}$	$1.11e+01_{9.40e-02}$
LP10_500	$1.55e+01_{6.02e-02}$	$4.12e+01_{1.91e+00}$	$1.38e+01_{1.15e-01}$
Media 500 ciudades	$1.46e+01_{4.79e-02}$	$3.02e+01_{5.92e-01}$	$1.35e+01_{1.12e-01}$
Media global	$5.82e+00_{4.05e-02}$	$2.20e+01_{5.49e-01}$	$4.76e+00_{1.03e-01}$

La tabla 4.2 muestra la media y la desviación estándar para el indicador distancia

generacional, GD, es decir, la distancia entre la aproximación del frente de Pareto proporcionada por los algoritmos y el frente de Pareto óptimo (o para ser más precisos, la aproximación que se ha hecho de éste según se describió en la sección 4.3). Como sabemos, un valor de $GD = 0$ nos indicaría que todos los elementos generados se encuentran en el frente de Pareto, por lo que la calidad de un algoritmo, en cuanto a GD se refiere, será mejor cuanto menor sea este indicador.

Los resultados para este indicador muestran un cambio claro al aumentar el número de ciudades. Para un número reducido de ciudades ($s = 10, 25$), AbYSS tiene, en media, una distancia generacional mejor, seguido, en este orden, de NSGA-II y SPEA2. No obstante, el algoritmo AbYSS deja de ser la mejor opción (en media) para un número de ciudades más elevado. Para $s = 50, 100$ es el algoritmo SPEA2 el que proporciona mejores resultados y para $s = 200, 500$ la mejor distancia generacional la consigue el algoritmo NSGA-II, superando también este último algoritmo a los otros dos en media global. A modo de ejemplo, se muestran en la figura 4.10 los diagramas de cajas del indicador distancia generacional para los tres algoritmos para todos los posibles números de ciudades del grupo de problemas LP05.

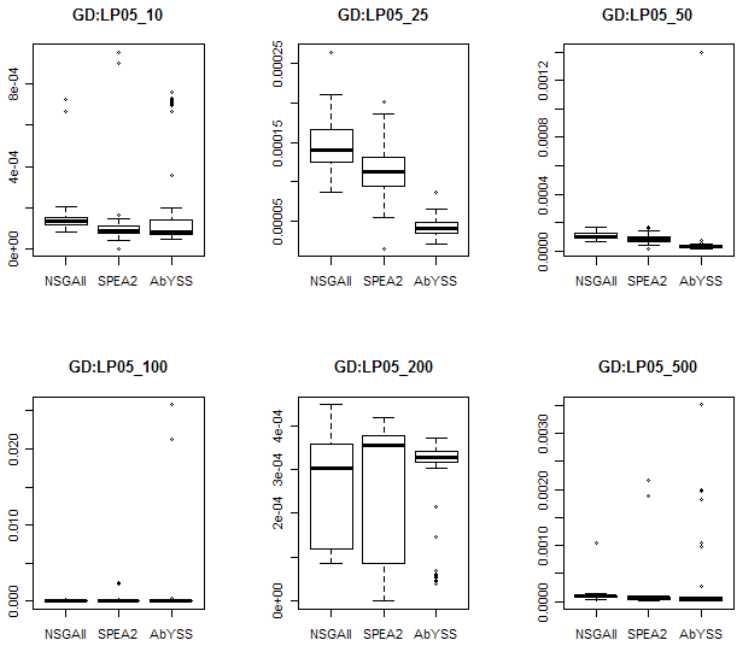

Figura 4.10: Distancia generacional. Problemas LP05

Tabla 4.2: GD. Media y desviación estándar

	NSGA-II	SPEA2	AbYSS
LP01_10	$4.40e-05_{1.6e-05}$	$3.75e-05_{5.3e-05}$	$7.72e-05_{1.7e-04}$
LP02_10	$7.50e-05_{2.5e-05}$	$4.91e-05_{2.0e-05}$	$2.41e-05_{7.5e-06}$
LP03_10	$6.25e-05_{2.6e-05}$	$1.84e-04_{2.6e-04}$	$2.56e-05_{1.3e-05}$
LP04_10	$9.83e-05_{3.0e-05}$	$7.68e-05_{2.9e-05}$	$4.33e-05_{9.0e-06}$
LP05_10	$1.47e-04_{8.2e-05}$	$1.07e-04_{1.2e-04}$	$2.00e-04_{2.4e-04}$
LP06_10	$9.61e-05_{2.1e-05}$	$1.02e-04_{3.6e-05}$	$7.12e-05_{5.2e-05}$
LP07_10	$4.16e-05_{2.1e-05}$	$5.97e-05_{8.5e-05}$	$3.31e-05_{2.7e-05}$
LP08_10	$2.32e-05_{1.0e-05}$	$1.57e-05_{1.2e-05}$	$1.79e-05_{2.8e-05}$
LP09_10	$1.06e-04_{1.0e-04}$	$1.70e-04_{1.9e-04}$	$1.05e-04_{1.1e-04}$
LP10_10	$5.08e-05_{1.3e-05}$	$7.69e-05_{1.1e-04}$	$1.99e-05_{6.3e-06}$
Media 10 ciudades	$7.45e-05_{3.44e-05}$	$8.79e-05_{9.15e-05}$	$6.17e-05_{6.63e-05}$
LP01_25	$1.15e-04_{1.1e-04}$	$1.40e-04_{2.4e-04}$	$1.46e-04_{6.5e-04}$
LP02_25	$7.66e-05_{2.4e-05}$	$4.87e-05_{1.9e-05}$	$4.08e-05_{1.6e-05}$
LP03_25	$1.03e-04_{2.5e-05}$	$1.23e-04_{8.6e-05}$	$4.55e-05_{1.4e-05}$
LP04_25	$2.25e-03_{4.3e-03}$	$3.33e-03_{5.9e-03}$	$1.60e-03_{3.6e-03}$
LP05_25	$1.45e-04_{2.9e-05}$	$1.13e-04_{2.9e-05}$	$4.25e-05_{1.1e-05}$
LP06_25	$9.16e-05_{2.2e-05}$	$7.13e-05_{2.4e-05}$	$2.20e-05_{5.5e-06}$
LP07_25	$7.43e-05_{2.7e-05}$	$6.47e-05_{2.5e-05}$	$3.06e-05_{8.6e-06}$
LP08_25	$1.19e-04_{2.6e-05}$	$9.37e-05_{3.6e-05}$	$4.09e-05_{1.5e-05}$
LP09_25	$6.49e-05_{2.1e-05}$	$5.75e-05_{3.2e-05}$	$6.32e-05_{1.1e-04}$
LP10_25	$1.37e-04_{2.9e-05}$	$1.16e-04_{3.6e-05}$	$5.36e-05_{1.6e-05}$
Media 25 ciudades	$3.18e-04_{4.61e-04}$	$4.16e-04_{6.43e-04}$	$2.09e-04_{4.45e-04}$
LP01_50	$7.57e-05_{1.6e-05}$	$6.59e-05_{5.1e-05}$	$2.62e-05_{8.0e-06}$
LP02_50	$1.25e-04_{2.6e-05}$	$8.58e-05_{2.2e-05}$	$1.21e-04_{4.3e-04}$
LP03_50	$1.24e-04_{2.9e-05}$	$7.79e-05_{2.5e-05}$	$3.15e-05_{1.1e-05}$
LP04_50	$7.32e-05_{2.0e-05}$	$5.11e-05_{2.3e-05}$	$1.67e-05_{6.0e-06}$
LP05_50	$1.10e-04_{2.1e-05}$	$8.56e-05_{2.4e-05}$	$4.88e-05_{1.4e-04}$
LP06_50	$8.38e-05_{2.6e-05}$	$9.35e-05_{6.2e-05}$	$6.00e-05_{4.3e-05}$
LP07_50	$5.11e-04_{2.1e-03}$	$5.73e-04_{2.9e-03}$	$3.18e-03_{4.8e-03}$
LP08_50	$1.64e-04_{1.6e-04}$	$1.36e-04_{1.7e-04}$	$1.71e-04_{9.7e-04}$
LP09_50	$1.20e-04_{3.2e-05}$	$9.27e-05_{2.7e-05}$	$3.94e-05_{1.3e-05}$
LP10_50	$1.93e-04_{4.2e-05}$	$1.73e-04_{4.8e-05}$	$1.56e-04_{7.0e-05}$
Media 50 ciudades	$1.58e-04_{2.47e-04}$	$1.43e-04_{3.35e-04}$	$3.85e-04_{6.49e-04}$
LP01_100	$1.23e-04_{2.4e-05}$	$6.87e-05_{1.6e-05}$	$3.68e-05_{8.7e-06}$
LP02_100	$8.52e-05_{2.2e-05}$	$7.24e-05_{5.9e-05}$	$2.42e-05_{8.9e-06}$
LP03_100	$1.60e-04_{2.3e-04}$	$1.11e-04_{2.4e-04}$	$4.31e-04_{6.5e-04}$
LP04_100	$9.23e-05_{2.5e-05}$	$5.69e-05_{2.4e-05}$	$2.83e-05_{8.0e-06}$
LP05_100	$1.24e-04_{2.1e-05}$	$1.28e-04_{3.2e-04}$	$5.16e-04_{3.3e-03}$
LP06_100	$1.40e-04_{3.5e-05}$	$9.46e-05_{3.3e-05}$	$5.29e-04_{1.8e-03}$
LP07_100	$1.11e-04_{3.2e-05}$	$8.07e-05_{3.1e-05}$	$2.80e-05_{1.0e-05}$
LP08_100	$1.82e-04_{9.5e-05}$	$1.47e-04_{1.6e-04}$	$2.26e-04_{3.8e-04}$
LP09_100	$1.35e-04_{2.5e-05}$	$1.04e-04_{3.8e-05}$	$4.21e-05_{1.5e-05}$
LP10_100	$1.15e-04_{2.3e-05}$	$2.09e-04_{1.4e-03}$	$1.63e-03_{4.3e-03}$
Media 100 ciudades	$1.27e-04_{5.32e-05}$	$1.07e-04_{2.32e-04}$	$3.49e-04_{1.05e-03}$
LP01_200	$1.12e-04_{2.8e-05}$	$8.45e-05_{2.9e-05}$	$2.38e-04_{1.0e-03}$
LP02_200	$1.11e-03_{8.0e-04}$	$1.12e-03_{9.2e-04}$	$9.22e-04_{2.0e-03}$
LP03_200	$1.31e-04_{2.3e-05}$	$7.84e-05_{1.8e-05}$	$3.64e-05_{8.9e-06}$
LP04_200	$1.52e-04_{2.8e-05}$	$1.00e-04_{2.7e-05}$	$8.92e-05_{3.0e-04}$
LP05_200	$2.52e-04_{1.2e-04}$	$2.68e-04_{1.4e-04}$	$2.96e-04_{9.2e-05}$
LP06_200	$1.50e-04_{2.5e-05}$	$1.03e-04_{2.9e-05}$	$1.05e-04_{2.0e-04}$
LP07_200	$1.20e-04_{2.9e-05}$	$6.67e-05_{1.7e-05}$	$1.23e-04_{9.6e-04}$
LP08_200	$9.90e-05_{2.4e-05}$	$2.54e-04_{1.1e-03}$	$1.67e-03_{2.8e-03}$
LP09_200	$7.99e-04_{1.3e-03}$	$1.27e-03_{2.0e-03}$	$1.88e-03_{1.9e-03}$
LP10_200	$1.14e-04_{2.5e-04}$	$1.36e-04_{3.2e-04}$	$1.48e-04_{3.1e-04}$
Media 200 ciudades	$3.04e-04_{2.63e-04}$	$3.48e-04_{4.60e-04}$	$5.51e-04_{9.57e-04}$
LP01_500	$1.18e-04_{2.4e-05}$	$9.14e-05_{2.6e-05}$	$1.60e-04_{5.5e-04}$
LP02_500	$2.24e-04_{9.9e-04}$	$1.87e-04_{1.1e-03}$	$1.91e-03_{3.6e-03}$
LP03_500	$1.51e-04_{3.3e-05}$	$9.34e-05_{1.1e-04}$	$3.12e-04_{4.8e-04}$
LP04_500	$1.58e-04_{3.0e-05}$	$6.36e-04_{3.1e-03}$	$2.39e-03_{5.2e-03}$
LP05_500	$1.08e-04_{9.8e-05}$	$1.06e-04_{2.8e-04}$	$2.45e-04_{6.0e-04}$
LP06_500	$1.31e-04_{2.4e-05}$	$8.06e-05_{2.1e-05}$	$3.49e-04_{1.2e-03}$
LP07_500	$7.16e-04_{1.0e-03}$	$8.48e-04_{1.5e-03}$	$1.15e-03_{1.7e-03}$
LP08_500	$9.46e-05_{1.9e-05}$	$7.61e-05_{1.8e-05}$	$9.69e-04_{1.7e-03}$
LP09_500	$2.36e-04_{2.9e-04}$	$2.59e-04_{5.2e-04}$	$3.35e-04_{6.0e-04}$
LP10_500	$2.24e-04_{2.0e-04}$	$2.15e-04_{2.2e-04}$	$2.36e-04_{2.4e-04}$
Media 500 ciudades	$2.16e-04_{2.71e-04}$	$2.59e-04_{6.90e-04}$	$8.06e-04_{1.59e-03}$
Media global	$1.99e-04_{2.22e-04}$	$2.27e-04_{4.09e-04}$	$3.93e-04_{7.92e-04}$

Los resultados obtenidos para el indicador spread, Δ, se encuentran en la tabla

4.3. Puesto que estamos interesados en obtener un conjunto de soluciones que se extienda por todo el frente Pareto-óptimo, nos interesarán los resultados que aporten un menor valor de este indicador, ya que Δ toma un valor igual a cero en el conjunto de soluciones no dominadas más amplio y uniformemente distribuido. Viendo los resultados, se corrobora lo que ya se apreciaba en las figuras de los frentes de Pareto mostrados anteriormente: el algoritmo AbYSS es, sin excepción, el que mejores resultados aporta para este indicador de calidad, seguido de NSGA-II y SPEA2 (SPEA2 es, en general, bastante peor que los otros dos algoritmos, aunque consigue mejorar a NSGA-II en tres casos puntuales: LP06_10, LP10_50 y LP09_500).

Al igual que se ha hecho para la distancia generacional, en la figura 4.11 se muestra los diagramas de cajas del indicador Δ para un grupo completo de problemas, en este caso para los problemas del grupo LP03.

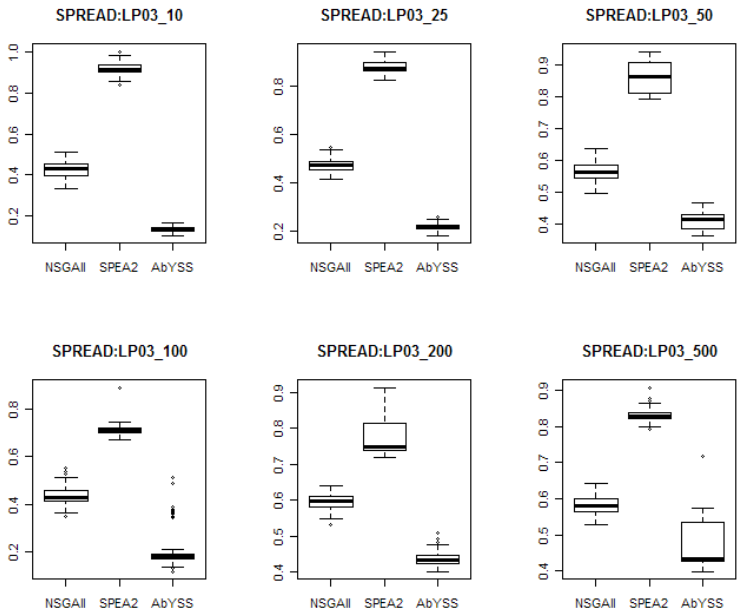

Figura 4.11: Spread. Problemas LP03

En la tabla 4.4 se muestra la media y la desviación estándar para el indicador de calidad hipervolumen HV. Recordemos que este indicador mide el volumen en el espacio objetivo cubierto por la aproximación del frente de Pareto, por lo que son preferibles los algoritmos que proporcionan mayores valores de HV. Los resultados son similares a los obtenidos para la distancia generacional, en el sentido

Tabla 4.3: SPREAD. Media y desviación estándar

	NSGA-II	SPEA2	AbYSS
LP01_10	$7.17e-01_{2.8e-02}$	$1.33e+00_{1.0e-01}$	$6.20e-01_{5.6e-02}$
LP02_10	$4.00e-01_{2.8e-02}$	$7.69e-01_{1.9e-02}$	$1.14e-01_{1.4e-02}$
LP03_10	$4.25e-01_{3.6e-02}$	$9.18e-01_{2.6e-02}$	$1.36e-01_{1.3e-02}$
LP04_10	$4.81e-01_{3.6e-02}$	$7.33e-01_{4.2e-02}$	$2.09e-01_{2.1e-02}$
LP05_10	$4.32e-01_{3.1e-02}$	$6.15e-01_{2.0e-02}$	$1.27e-01_{1.8e-02}$
LP06_10	$6.25e-01_{2.5e-02}$	$5.62e-01_{1.3e-02}$	$4.86e-01_{1.4e-02}$
LP07_10	$5.36e-01_{4.1e-02}$	$1.22e+00_{3.7e-02}$	$4.29e-01_{1.0e-01}$
LP08_10	$4.17e-01_{3.1e-02}$	$1.03e+00_{1.4e-02}$	$1.48e-01_{1.4e-02}$
LP09_10	$7.97e-01_{2.6e-02}$	$1.02e+00_{4.4e-02}$	$7.35e-01_{5.1e-02}$
LP10_10	$6.09e-01_{8.4e-02}$	$1.21e+00_{3.1e-02}$	$5.26e-01_{1.2e-01}$
Media 10 ciudades	$5.44e-01_{3.66e-02}$	$9.41e-01_{3.46e-02}$	$3.53e-01_{4.21e-02}$
LP01_25	$5.76e-01_{2.7e-02}$	$1.03e+00_{5.7e-02}$	$4.05e-01_{4.2e-02}$
LP02_25	$4.31e-01_{3.0e-02}$	$9.32e-01_{1.3e-02}$	$1.98e-01_{1.3e-02}$
LP03_25	$4.72e-01_{2.7e-02}$	$8.79e-01_{2.3e-02}$	$2.16e-01_{1.4e-02}$
LP04_25	$5.75e-01_{1.3e-01}$	$9.36e-01_{9.1e-02}$	$4.22e-01_{1.6e-01}$
LP05_25	$4.06e-01_{3.1e-02}$	$4.51e-01_{2.4e-02}$	$9.86e-02_{1.3e-02}$
LP06_25	$4.26e-01_{3.5e-02}$	$6.29e-01_{2.3e-02}$	$1.35e-01_{1.8e-02}$
LP07_25	$4.28e-01_{3.0e-02}$	$9.02e-01_{1.4e-02}$	$1.59e-01_{1.2e-02}$
LP08_25	$4.23e-01_{3.9e-02}$	$6.56e-01_{2.0e-02}$	$1.34e-01_{1.6e-02}$
LP09_25	$5.03e-01_{2.5e-02}$	$1.09e+00_{1.2e-02}$	$3.17e-01_{8.3e-02}$
LP10_25	$4.12e-01_{3.1e-02}$	$6.23e-01_{1.8e-02}$	$1.30e-01_{1.4e-02}$
Media 25 ciudades	$4.65e-01_{4.05e-02}$	$8.13e-01_{2.95e-02}$	$2.21e-01_{3.85e-02}$
LP01_50	$7.73e-01_{1.8e-02}$	$1.26e+00_{3.7e-02}$	$6.81e-01_{4.7e-02}$
LP02_50	$5.19e-01_{2.8e-02}$	$7.86e-01_{1.7e-02}$	$3.33e-01_{1.6e-02}$
LP03_50	$5.67e-01_{3.1e-02}$	$8.61e-01_{4.9e-02}$	$4.13e-01_{2.5e-02}$
LP04_50	$5.48e-01_{2.2e-02}$	$9.64e-01_{3.2e-02}$	$3.68e-01_{1.3e-02}$
LP05_50	$4.96e-01_{3.7e-02}$	$7.96e-01_{1.8e-02}$	$3.37e-01_{8.0e-02}$
LP06_50	$5.26e-01_{3.3e-02}$	$1.06e+00_{2.0e-02}$	$3.43e-01_{7.3e-02}$
LP07_50	$5.87e-01_{4.5e-02}$	$1.06e+00_{2.1e-02}$	$5.24e-01_{1.4e-01}$
LP08_50	$5.39e-01_{2.8e-02}$	$7.78e-01_{3.2e-02}$	$3.36e-01_{1.9e-02}$
LP09_50	$4.07e-01_{3.5e-02}$	$6.71e-01_{1.7e-02}$	$1.22e-01_{1.6e-02}$
LP10_50	$4.06e-01_{2.9e-02}$	$3.92e-01_{2.5e-02}$	$1.14e-01_{1.3e-02}$
Media 50 ciudades	$5.37e-01_{3.06e-02}$	$8.63e-0_{2.68e-02}$	$3.57e-01_{4.42e-02}$
LP01_100	$5.93e-01_{2.7e-02}$	$7.59e-01_{3.9e-02}$	$4.31e-01_{2.2e-02}$
LP02_100	$7.37e-01_{2.6e-02}$	$1.13e+00_{9.0e-02}$	$6.74e-01_{1.7e-02}$
LP03_100	$4.38e-01_{3.8e-02}$	$7.11e-01_{2.3e-02}$	$2.03e-01_{7.3e-02}$
LP04_100	$5.93e-01_{2.0e-02}$	$1.05e+00_{1.9e-02}$	$4.71e-01_{2.4e-02}$
LP05_100	$5.30e-01_{2.4e-02}$	$7.29e-01_{2.4e-02}$	$3.61e-01_{5.9e-02}$
LP06_100	$4.53e-01_{2.9e-02}$	$6.35e-01_{1.7e-02}$	$2.55e-01_{1.4e-01}$
LP07_100	$4.87e-01_{2.9e-02}$	$8.12e-01_{1.4e-02}$	$2.53e-01_{9.9e-03}$
LP08_100	$5.51e-01_{3.0e-02}$	$7.38e-01_{4.9e-02}$	$3.90e-01_{3.3e-02}$
LP09_100	$4.14e-01_{3.0e-02}$	$5.62e-01_{2.1e-02}$	$1.08e-01_{1.4e-02}$
LP10_100	$6.46e-01_{3.6e-02}$	$7.20e-01_{8.8e-02}$	$4.79e-01_{1.3e-01}$
Media 100 ciudades	$5.44e-01_{2.89e-02}$	$7.85e-01_{3.84e-02}$	$3.63e-01_{5.22e-02}$
LP01_200	$6.20e-01_{3.4e-02}$	$1.05e+00_{2.2e-02}$	$5.25e-01_{5.8e-02}$
LP02_200	$7.35e-01_{2.8e-02}$	$7.37e-01_{4.4e-02}$	$6.03e-01_{9.2e-02}$
LP03_200	$5.97e-01_{2.2e-02}$	$7.79e-01_{6.1e-02}$	$4.37e-01_{2.1e-02}$
LP04_200	$4.86e-01_{2.8e-02}$	$6.52e-01_{1.9e-02}$	$2.90e-01_{1.9e-02}$
LP05_200	$5.13e-01_{2.8e-02}$	$7.54e-01_{1.8e-02}$	$3.26e-01_{1.1e-02}$
LP06_200	$4.30e-01_{2.8e-02}$	$5.41e-01_{1.9e-02}$	$1.66e-01_{4.2e-02}$
LP07_200	$6.78e-01_{2.0e-02}$	$8.58e-01_{4.2e-02}$	$5.57e-01_{2.5e-02}$
LP08_200	$7.07e-01_{2.4e-02}$	$9.99e-01_{5.2e-02}$	$5.97e-01_{4.4e-02}$
LP09_200	$6.47e-01_{3.0e-02}$	$9.59e-01_{2.6e-02}$	$5.58e-01_{5.9e-02}$
LP10_200	$8.67e-01_{5.3e-02}$	$1.25e+00_{1.5e-01}$	$7.98e-01_{7.8e-02}$
Media 200 ciudades	$6.28e-01_{2.95e-02}$	$8.58e-01_{4.53e-02}$	$4.86e-01_{4.49e-02}$
LP01_500	$5.64e-01_{2.6e-02}$	$8.20e-01_{1.5e-02}$	$4.33e-01_{9.2e-02}$
LP02_500	$6.25e-01_{3.1e-02}$	$7.75e-01_{2.8e-02}$	$6.17e-01_{1.3e-01}$
LP03_500	$5.80e-01_{2.5e-02}$	$8.31e-01_{1.8e-02}$	$4.79e-01_{7.0e-02}$
LP04_500	$5.03e-01_{2.5e-02}$	$6.29e-01_{4.9e-02}$	$3.71e-01_{1.7e-01}$
LP05_500	$7.42e-01_{2.2e-02}$	$1.08e+00_{2.9e-02}$	$6.99e-01_{3.0e-02}$
LP06_500	$5.87e-01_{3.2e-02}$	$8.47e-01_{3.2e-02}$	$4.65e-01_{3.9e-02}$
LP07_500	$7.72e-01_{1.9e-02}$	$9.46e-01_{2.8e-02}$	$7.18e-01_{2.4e-02}$
LP08_500	$6.65e-01_{2.5e-02}$	$8.13e-01_{1.8e-02}$	$6.20e-01_{1.2e-01}$
LP09_500	$4.41e-01_{3.3e-02}$	$4.38e-01_{2.5e-02}$	$2.09e-01_{5.8e-02}$
LP10_500	$6.40e-01_{3.2e-02}$	$7.82e-01_{2.7e-02}$	$5.75e-01_{3.9e-02}$
Media 500 ciudades	$6.12e-01_{2.70e-02}$	$7.96e-01_{2.69e-02}$	$5.19e-01_{7.72e-02}$
Media global	$5.55e-01_{3.22e-02}$	$8.42e-01_{3.36e-02}$	$3.83e-01_{4.98e-02}$

de que hay un cambio muy claro al aumentar el número de ciudades contempladas en los problemas. Para $s = 10, 25$, el algoritmo AbYSS cubre un mayor volumen del espacio objetivo con las soluciones no dominadas que encuentra, pero a partir de un número de ciudades igual a 50 es NSGA-II el que, en media, proporciona los mejores resultados. El algoritmo SPEA2 también parece mejorar su comportamiento al aumentar el tamaño de los problemas. Se muestran en la figura 4.12 los diagramas de cajas del indicador HV para los problemas del grupo LP08.

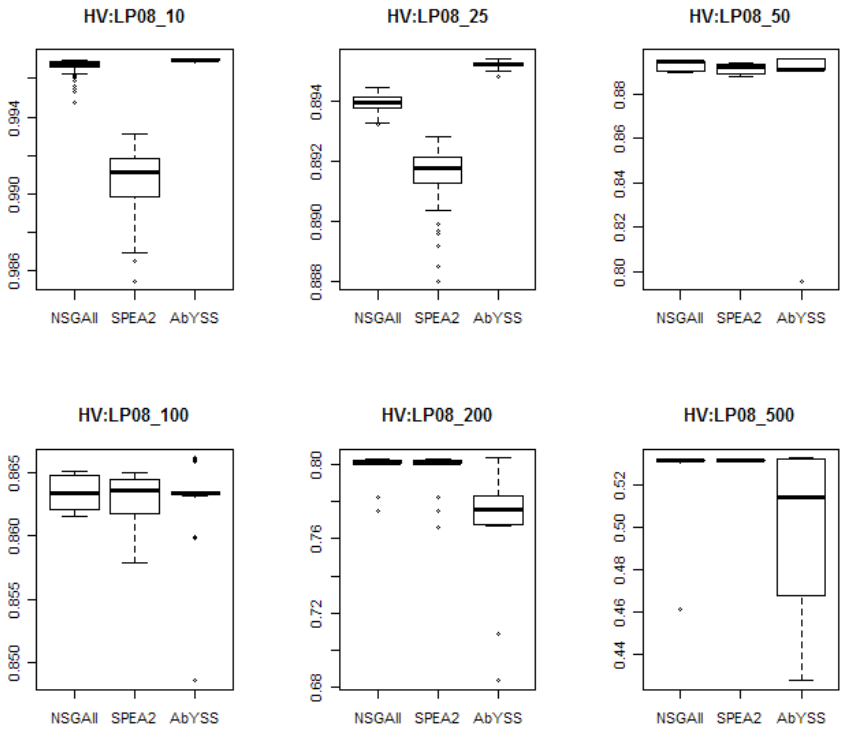

Figura 4.12: Hipervolumen. Problemas LP08

Tabla 4.4: HV. Media y desviación estándar

	NSGA-II	SPEA2	AbYSS
LP01_10	$9.93e-01_{7.9e-05}$	$9.91e-01_{3.7e-04}$	$9.93e-01_{1.0e-04}$
LP02_10	$9.55e-01_{1.7e-04}$	$9.52e-01_{5.4e-04}$	$9.56e-01_{4.6e-05}$
LP03_10	$9.90e-01_{2.0e-04}$	$9.83e-01_{2.5e-03}$	$9.91e-01_{5.3e-05}$
LP04_10	$9.52e-01_{2.3e-04}$	$9.48e-01_{1.2e-03}$	$9.53e-01_{8.6e-05}$
LP05_10	$8.58e-01_{6.3e-04}$	$8.56e-01_{7.1e-04}$	$8.59e-01_{1.5e-03}$
LP06_10	$9.38e-01_{1.4e-04}$	$9.37e-01_{3.6e-04}$	$9.39e-01_{2.6e-04}$
LP07_10	$9.88e-01_{1.6e-04}$	$9.85e-01_{9.3e-04}$	$9.88e-01_{6.1e-05}$
LP08_10	$9.97e-01_{3.7e-04}$	$9.91e-01_{1.6e-03}$	$9.97e-01_{2.4e-05}$
LP09_10	$9.47e-01_{2.0e-04}$	$9.43e-01_{1.1e-03}$	$9.45e-01_{5.0e-03}$
LP10_10	$9.81e-01_{4.0e-04}$	$9.78e-01_{8.5e-04}$	$9.81e-01_{4.8e-04}$
Media 10 ciudades	$9.60e-01_{2.58e-04}$	$9.56e-01_{1.02e-03}$	$9.60e-01_{7.61e-04}$
LP01_25	$9.56e-01_{1.3e-04}$	$9.54e-01_{7.8e-04}$	$9.56e-01_{1.7e-03}$
LP02_25	$9.54e-01_{1.7e-04}$	$9.50e-01_{7.2e-04}$	$9.54e-01_{9.1e-05}$
LP03_25	$9.29e-01_{2.0e-04}$	$9.24e-01_{1.7e-03}$	$9.30e-01_{1.1e-04}$
LP04_25	$8.67e-01_{2.5e-04}$	$8.66e-01_{4.8e-04}$	$8.68e-01_{1.4e-04}$
LP05_25	$7.77e-01_{2.8e-04}$	$7.77e-01_{3.5e-04}$	$7.79e-01_{7.8e-05}$
LP06_25	$8.89e-01_{2.3e-04}$	$8.87e-01_{5.3e-04}$	$8.91e-01_{5.7e-05}$
LP07_25	$9.70e-01_{1.6e-04}$	$9.66e-01_{9.1e-04}$	$9.70e-01_{5.2e-05}$
LP08_25	$8.94e-01_{2.6e-04}$	$8.92e-01_{8.7e-04}$	$8.95e-01_{8.2e-05}$
LP09_25	$9.70e-01_{1.1e-04}$	$9.67e-01_{8.9e-04}$	$9.71e-01_{1.2e-03}$
LP10_25	$8.79e-01_{2.9e-04}$	$8.77e-01_{9.5e-04}$	$8.81e-01_{8.3e-05}$
Media 25 ciudades	$9.09e-01_{2.08e-04}$	$9.06e-01_{8.18e-04}$	$9.10e-01_{3.59e-04}$
LP01_50	$8.53e-01_{1.9e-04}$	$8.53e-01_{4.1e-04}$	$8.50e-01_{2.4e-02}$
LP02_50	$8.10e-01_{2.2e-04}$	$8.09e-01_{4.3e-04}$	$8.11e-01_{4.1e-03}$
LP03_50	$8.67e-01_{5.1e-04}$	$8.66e-01_{1.0e-03}$	$8.68e-01_{5.7e-04}$
LP04_50	$9.48e-01_{1.4e-04}$	$9.46e-01_{4.8e-04}$	$9.49e-01_{4.0e-05}$
LP05_50	$8.93e-01_{3.1e-04}$	$8.92e-01_{5.0e-04}$	$8.94e-01_{1.3e-03}$
LP06_50	$9.26e-01_{9.7e-04}$	$9.23e-01_{1.5e-03}$	$9.25e-01_{2.2e-03}$
LP07_50	$8.78e-01_{9.1e-03}$	$8.77e-01_{8.9e-03}$	$8.62e-01_{2.7e-02}$
LP08_50	$8.92e-01_{2.2e-03}$	$8.91e-01_{2.2e-03}$	$8.92e-01_{9.9e-03}$
LP09_50	$8.99e-01_{2.4e-04}$	$8.97e-01_{6.6e-04}$	$9.01e-01_{8.1e-05}$
LP10_50	$7.05e-01_{5.3e-04}$	$7.05e-01_{6.0e-04}$	$7.07e-01_{6.3e-04}$
Media 50 ciudades	$8.67e-01_{1.44e-03}$	$8.66e-01_{1.67e-03}$	$8.66-01_{6.98e-03}$
LP01_100	$7.98e-01_{1.8e-04}$	$7.98e-01_{1.5e-04}$	$7.99e-01_{5.6e-05}$
LP02_100	$8.69e-01_{5.5e-03}$	$8.68e-01_{5.3e-03}$	$8.62e-01_{8.3e-03}$
LP03_100	$8.45e-01_{2.4e-03}$	$8.43e-01_{1.9e-03}$	$8.43e-01_{6.1e-03}$
LP04_100	$9.03e-01_{1.8e-04}$	$9.02e-01_{4.1e-04}$	$9.03e-01_{5.4e-04}$
LP05_100	$6.29e-01_{2.0e-04}$	$6.27e-01_{6.3e-03}$	$6.28e-01_{1.8e-02}$
LP06_100	$7.63e-01_{2.4e-04}$	$7.63e-01_{4.4e-04}$	$7.50e-01_{3.8e-02}$
LP07_100	$8.44e-01_{2.1e-04}$	$8.43e-01_{5.6e-04}$	$8.46e-01_{8.3e-05}$
LP08_100	$8.63e-01_{1.3e-03}$	$8.63e-01_{1.5e-03}$	$8.63e-01_{2.9e-03}$
LP09_100	$8.45e-01_{2.5e-04}$	$8.43e-01_{5.8e-04}$	$8.46e-01_{9.4e-05}$
LP10_100	$8.36e-01_{1.8e-04}$	$8.35e-01_{1.1e-02}$	$8.23e-01_{3.7e-02}$
Media 100 ciudades	$8.20e-01_{1.06e-03}$	$8.19e-01_{2.81e-03}$	$8.16e-01_{1.11e-02}$
LP01_200	$8.14e-01_{1.9e-03}$	$8.12e-01_{2.9e-03}$	$8.10e-01_{1.4e-02}$
LP02_200	$6.99e-01_{2.3e-03}$	$6.99e-01_{3.0e-03}$	$6.94e-01_{1.8e-02}$
LP03_200	$8.42e-01_{1.7e-04}$	$8.42e-01_{3.1e-04}$	$8.44e-01_{6.0e-05}$
LP04_200	$7.70e-01_{2.5e-04}$	$7.70e-01_{4.7e-04}$	$7.71e-01_{2.0e-03}$
LP05_200	$7.19e-01_{6.8e-04}$	$7.19e-01_{7.0e-04}$	$7.20e-01_{3.5e-04}$
LP06_200	$7.44e-01_{2.8e-04}$	$7.44e-01_{4.1e-04}$	$7.45e-01_{4.7e-03}$
LP07_200	$7.21e-01_{1.3e-03}$	$7.21e-01_{1.3e-03}$	$7.20e-01_{8.8e-03}$
LP08_200	$7.97e-01_{8.6e-03}$	$7.97e-01_{9.7e-03}$	$7.78e-01_{2.1e-02}$
LP09_200	$7.40e-01_{6.1e-03}$	$7.39e-01_{6.7e-03}$	$7.31e-01_{1.8e-02}$
LP10_200	$6.60e-01_{1.4e-02}$	$6.54e-01_{1.8e-02}$	$6.44e-01_{2.0e-02}$
Media 200 ciudades	$7.51e-01_{3.56e-03}$	$7.50e-01_{4.35e-03}$	$7.46e-01_{1.07e-02}$
LP01_500	$6.49e-01_{1.9e-04}$	$6.48e-01_{5.1e-04}$	$6.42e-01_{3.8e-02}$
LP02_500	$5.63e-01_{6.5e-03}$	$5.63e-01_{6.5e-03}$	$5.38e-01_{3.3e-02}$
LP03_500	$7.13e-01_{2.1e-03}$	$7.12e-01_{3.6e-03}$	$7.03e-01_{1.1e-02}$
LP04_500	$6.95e-01_{2.8e-04}$	$6.94e-01_{8.3e-03}$	$6.88e-01_{1.9e-02}$
LP05_500	$7.27e-01_{7.3e-03}$	$7.25e-01_{9.4e-03}$	$7.09e-01_{2.3e-02}$
LP06_500	$8.05e-01_{4.5e-03}$	$8.04e-01_{4.9e-03}$	$7.99e-01_{1.7e-02}$
LP07_500	$6.99e-01_{7.9e-03}$	$6.99e-01_{1.0e-02}$	$6.62e-01_{4.6e-02}$
LP08_500	$5.31e-01_{7.0e-03}$	$5.31e-01_{1.9e-04}$	$5.03e-01_{3.2e-02}$
LP09_500	$6.33e-01_{2.4e-03}$	$6.33e-01_{4.9e-03}$	$6.32e-01_{8.7e-03}$
LP10_500	$5.70e-01_{3.5e-03}$	$5.69e-01_{5.1e-03}$	$5.64e-01_{9.1e-03}$
Media 500 ciudades	$6.59e-01_{4.17e-03}$	$6.58e-01_{5.34e-03}$	$6.44e-01_{2.37e-02}$
Media global	$8.27e-01_{1.78e-03}$	$8.26e-01_{2.67e-03}$	$8.24e-01_{8.93e-03}$

5

Conclusiones

En este trabajo se ha presentado un nuevo modelo para la localización de centros semi-repulsivos, entendiéndose como tales, aquellas instalaciones que pueden resultar atractivas para determinados colectivos, pero no deseadas por otros. Se han considerado instalaciones cuyos efectos nocivos no ponen en peligro la vida de la gente, al menos directamente o de manera repentina, como pueden ser centrales eléctricas, vertederos para residuos sólidos urbanos o un aeropuerto.

El modelo se traduce en un problema de optimización biobjetivo, en el que se considera como primer objetivo uno de minimización de costes de transporte y, como segundo objetivo, uno de minimización de la repulsión global hacia el centro. Además, se han tenido en cuenta preocupaciones medioambientales a través de la definición de áreas protegidas donde no se admite la ubicación de las instalaciones. Otras características importantes son que alrededor de cada ciudad hay también una región prohibida para evitar la localización muy cerca de ellas y que el conjunto factible puede tener cualquier forma, siendo el único requisito el ser capaz de especificarlo a través de un conjunto de restricciones analíticas.

La resolución de este modelo se ha llevado a cabo a través de tres algoritmos genéticos multiobjetivo de referencia: NSGA-II, un algoritmo elitista basado en un enfoque de ordenación rápida de soluciones no dominadas, SPEA2, una versión mejorada de su antecesor SPEA que emplea una estrategia de asignación de *fitness* más avanzada y nuevas técnicas para el truncamiento del archivo y la selección

basadas en densidad, y AbYSS, un algoritmo evolutivo basado en búsqueda dispersa que combina además aspectos de los algoritmos NSGA-II y SPEA2.

Tras estudiar estos tres algoritmos evolutivos, y haciendo uso de la plataforma jMetal, se han resuelto una amplia variedad de problemas de localización de centros semi-repulsivos. Se han estudiado sus tiempos de ejecución y los indicadores de calidad GD (distancia generacional), Δ (spread) y HV (hipervolumen) para conocer la proximidad al frente de Pareto y la diversidad de las soluciones obtenidas. Además, para una mejor interpretación de los problemas, se ha realizado la representación gráfica de la solución de algunos de los problemas, tanto en el espacio origen como en el imagen, a través de la herramienta Tcl-Tk.

Los resultados obtenidos muestran cómo el algoritmo AbYSS supera a NSGA-II y SPEA2 en cuanto a tiempo de ejecución se refiere y a diversidad de sus soluciones (indicador Δ). Por lo que respecta a la distancia generacional GD (distancia entre la aproximación del frente de Pareto ofrecido por los algoritmos y el frente de Pareto óptimo) y al hipervolumen HV (volumen en el espacio objetivo cubierto por el conjunto solución) puede apreciarse un cambio en el comportamiento de los algoritmos a medida que aumenta el tamaño de los problemas. AbYSS obtiene, en general, mejores resultados para problemas pequeños, mientras que los algoritmos SPEA2 y NSGA-II consiguen superarlo en problemas de mayor tamaño. El algoritmo NSGA-II es el que consigue mejores resultados, en media, para los indicadores GD y HV, seguido de SPEA2 y AbYSS.

Bibliografía

[1] Brimberg, J., Love, R.F. and Walker, J.H. (1995). *The effect of axis rotation on distance estimation*. European Journal of Operational Research 80(2), 357-364.

[2] De Jong, K.A. (1975). *An analysis of the behavior of a class of genetic adaptive systems*. Doctoral dissertation, University of Michigan, Ann Arbor, Michigan. `http://hdl.handle.net/2027.42/4507`

[3] Deb, K. and Agrawal, R.B. (1995). *Simulated binary crossover for continuous search space*. Complex Systems 9, 115-148. Kanpur, India.

[4] Deb, K. and Goyal, M. (1996). *A Combined Genetic Adaptive Search (GeneAS) for Engineering Design*. Computer Science and Informatics 26(4), 30-45.

[5] Deb, K. et al. (2002). *A fast and elitist multiobjective genetic algorithm: NSGA-II*. IEEE Transactions on Evolutionary Computation 6(2), 182-197.

[6] Durillo, J.J., Nebro, A.J. and Alba, E. (2010). *The jMetal framework for multi-objective optimization: Design and architecture*, CEC 2010, Lecture Notes in Computer Science 5467, 4138-4325.

[7] Durillo, J.J. and Nebro, A.J. (2011). *jmetal: A java framework for multi-objective optimization*. Advances in Engineering Software 42(10), 760-771.

[8] Edgeworth, F.Y. (1987, edición original de 1881). *Mathematical Psychics*. London: University Microfilms International.

[9] Fernández, J., Fernández, P. and Pelegrín, B. (2000). *A continuous location model for siting a non-noxious undesirable facility within a geographical region*. European Journal of Operational Research 121(2), 259-274.

[10] Fernández, J., Fernández, P. and Pelegrín, B. (2002) *Estimating actual distances by norm functions: a comparison between the* $l_{k,p,\theta}-norm$ *and the* $l_{b_1,b_2,\theta}-norm$. Computers and Operations Research 29(6), 609-623.

[11] Glover, F.W. and Kochenberger, G.A. (2003). *Handbook of Metaheuristics*, Int. Series in Operations Research and Management Sciences. Massachusetts: Kluwer Academic Publishing.

[12] Goldberg, D.E. (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning*. Massachusetts: Addison-Wesley.

[13] Holland, J.H. (1975, reeditado por MIT Press en 1992). *Adaptation in Natural and Artificial Systems*. Michigan: University of Michigan Press.

[14] Knowles, J. and Corne, D. (1999). *The pareto archived evolution strategy: A new baseline algorithm for multiobjective optimization*. Congress on Evolutionary Computation, 9-105. Piscataway, New Jersey: IEEE Press.

[15] Miettinen, K.M. (2002). *Nonlinear multiobjective optimization*. Massachusetts: Kluwer Academic Publishing.

[16] Molina, J. et al. (2007). *SSPMO: A scatter tabu search procedure for non-linear multiobjective optimization*, INFORMS Journal on Computing 19(1), 91-100.

[17] Nebro, A.J. et al. (2008). *AbYSS: Adapting Scatter Search to Multiobjective Optimization*. IEEE Transactions on Evolutionary Computation 12(4), 439-457.

[18] Pareto, V. (1964, primera edición en 1896). *Cours d'Economie Politique*. Genève: Libraire Droz.

[19] Pareto, V. (1971, edición original en francés de 1927). *Manual of Political Economy*. London: The MacMillan Press Ltd.

[20] Reeves, C.R. (1993). *Using genetic algorithms with small populations*, In S. Forrest (ed.) Proceedings of 5th International Conference on Genetic Algorithms, 92-99. San Mateo, California: Morgan Kaufmann.

[21] Silverman, B.W. (1986). *Density estimation for statistics and data analysis*. London: Chapman and Hall.

[22] Srinivas, N. and Deb, K. (1995). *Multiobjective function optimization using nondominated sorting genetic algorithms*. Evolutionary Computation 2(3), 221-248.

[23] Van Veldhuizen, D.A. and Lamont, G.B. (1998). *Multiobjective Evolutionary Algorithm Research: A History and Analysis*, Dept. Elec. Comput. Eng., Graduate School of Eng., Air Force Inst. Technol., Wright-Patterson, AFB, OH, Tech. Rep. TR-98-03, 1998. `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.35.8924&rep=rep1&type=pdf`

[24] Weber, A. (1909). *Über den Standort der Insdustrien*. Tübingen. Traducción de Friederich, C.J. (1929). *Theory of the location industries*. Chicago: University of Chicago Press.

[25] Zitzler, E. and Thiele, L. (1999). *Multiobjective Evolutionary Algorithms: A Comparative Case Study and the Strength Pareto Approach*. IEEE Transactions on Evolutionary Computation 3(4), 257-271.

[26] Zitzler, E., Deb, K. and Thiele, L. (2000). *Comparison of multiobjective evolutionary algorithms: Empirical results*. Evolutionary Computation 8(2), 173-195.

[27] Zitzler, E., Laumanns, M. and Thiele, L. (2001). *SPEA2: Improving the strength pareto evolutionary algorithm*. Computer Engineering and Networks Laboratory (TIK), Swiss Federal Institute of Technology (ETH), Zurich, Switzerland, `http://www.tik.ee.ethz.ch/sop/publicationListFiles/zlt2001a.pdf`

Apéndice: documentos adjuntos

Junto a esta memoria se incluye un CD con todos aquellos ficheros que han sido necesarios para la generación y resolución de los problemas estudiados, así como para su representación gráfica. El CD contiene:

1. Manual de usuario de jMetal.

2. Carpeta *jmetal*: estructura de paquetes de jMetal

   core: contiene los ingredientes básicos para ser utilizados por las metaheurísticas desarrolladas en jMetal. Las principales clases de este paquete son *Algorithm*, *Operator*, *Problem*, *Variable*, *Solution*, *SolutionSet* y *SolutionType*.

   encodings: contiene las representaciones básicas de variables y tipos de soluciones incluidos en la plataforma.

   experiments: contiene un conjunto de clases destinadas a la realización de estudios típicos en la optimización multiobjetivo. Los ficheros implementados para resolver los problemas de localización de un centro semi-repulsivo se encuentran en este paquete:
   GenerateProblem.java (carpeta *util*)
   Location.java
   GenerateParetoFront.java (carpeta *util*)
   LocationQuality.java

   metaheuristics: contiene las metaheurísticas implementadas en jMetal.

   operators: contiene diferentes tipos de objeto operador, incluyendo operadores de cruce, mutación, selección y operadores de búsqueda local.

   problems: todos los problemas disponibles en jMetal. Los 60 problemas estudiados de localización de un centro semi-repulsivo se encuentran en este paquete.

   qualityIndicator: indicadores de calidad para evaluar el desempeño de las metaheurísticas multiobjetivo.

   util: clases con funcionalidades utilizadas por otros paquetes, como un generador de números pseudoaleatorio, comparadores, etc.

3. Carpeta *ParetoFront*: aproximaciones al frente de Pareto óptimo de los 60 problemas estudiados.

4. Carpeta *SalidaAlgoritmos*: resultados de las 100 ejecuciones de los 60 problemas tratados para los algoritmos NSGA-II, SPEA2 y AbYSS.

5. Carpeta *Tcl-Tk*: ficheros necesarios para utilizar la herramienta Tcl-Tk. Contiene el ejecutable *genera_fichero.exe*, el fichero *escala.tcl* y el fichero *COLOR*. A modo de ejemplo, se incluye la carpeta *ficherosLP01* con todos los ficheros *datos*, *VAR*, *FUN*, *espacio_origen* y *frente_pareto* de todos los problemas del grupo LP01.

6. Fichero *Codigo_Programas_Implementados.pdf*: códigos de los ficheros *GenerateProblem.java*, *Location.java*, *GenerateParetoFront.java*, *LocationQuality.java*, *genera_fichero.cpp* y *escala.tcl*.