UNIVERSIDAD DE EXTREMADURA

Escuela Politécnica

Grado en Ingeniería Informática en Ingeniería De Computadores

Trabajo Fin de Grado

Optimización de Requisitos Software mediante un Algoritmo Evolutivo Multiobjetivo basado en Inteligencia de Enjambre

UNIVERSIDAD DE EXTREMADURA

Escuela Politécnica
Grado en Ingeniería Informática en Ingeniería de Computadores

Trabajo Fin de Grado

Optimización de Requisitos Software mediante un Algoritmo Evolutivo Multiobjetivo basado en Inteligencia de Enjambre

Autor: Jesús Chávez Águedo

Tutor: Miguel Ángel Vega Rodríguez

ÍNDICE GENERAL DE CONTENIDOS

1. RESUMEN

Lo que vamos a ver en las siguientes páginas, va a ser un desarrollo sobre cómo se optimizan los recursos softwares mediante intentando cumplir dos objetivos, conseguir la máxima satisfacción con el mínimo esfuerzo posible. Para poder cumplir estos objetivos, utilizaremos algoritmos con inteligencia de enjambre.

Para poder entender todo, debemos explicar cada concepto y cada subapartado detenidamente. Hay muchos conceptos que debemos entender antes de comparar resultados y saber cómo se ha desarrollado el proyecto.

Explicaremos términos como la dominancia, sobre cómo un individuo domina a otro y ver qué características han de cumplirse para que ocurra esta dominancia. También veremos la distancia crawding y los diferentes frentes de paretos que existen, esto último se apoya en la dominancia.

El algoritmo evolutivo que usamos es el Algoritmo de las Ranas Saltarinas (Shuffled Frog Leaping Algorithm – SFLA). Es el algoritmo que se propuso desde un principio, además es sencillo de entender y da bastantes buenos resultados en comparación con el resto de algoritmos evolutivos. También decidimos utilizar este algoritmo porque es con el que vamos a hacer la comparación directa.

Los resultado que hemos obtenido son competentes, varian con los resultados que deseamos, se comparará los resultados obtenidos más adelante, ya que hay varias cosas que decir sobre los resultados. También se compararán con otros algoritmos, para saber cuál es más óptimo.

Todo esto lo veremos a lo largo de la documentación de este documento.

2. INTRODUCCIÓN

Lo que vamos a ver en esta sección va a ser cómo va a estar estructurada nuestra documentación, se divide en varios apartados principales.

2.1 Estructura de la documentación

Primeramente la documentación está dividida en cuatro grandes apartados. La primera parte es en la que nos encontramos, es la **Introducción** donde nos encontraremos toda la información a priori que debemos saber antes de introducirnos más profundamente en la documentación, además esta parte explica brevemente qué podemos encontrarnos en la documentación, para que no haya ningún mal entendido.

La segunda parte es bastante más amplia que la primera, llamada **Antecedentes**, esta parte se centra en explicarnos detalladamente todos los conceptos que debemos saber antes de explicar cómo se ha desarrollado y resuelto nuestro problema. Explicaremos más detalladamente esta parte, ya que hay varios puntos en los que debemos hacer hincapié.

El primer apartado de esta sección llamado **Requisitos Softwares** nos explica generalmente sobre de qué va a ir nuestro proyecto, los siguientes apartados explican otros conceptos como la **optimización multiobjetivo** o el uso de **algoritmos evolutivos**

La segunda parte llamada **Material y Método** nos indica qué metodología hemos empleado para resolver el problema propuesto, además de los recursos que se han utilizado durante la implementación. Especialmente nos pararemos en la metodología empleada, ya que es muy importante entenderla porque después la compararemos.

La penúltima parte son los **Resultados y Discusión** en este apartado veremos todos los resultado obtenidos de nuestro algoritmo, a raíz de esos

resultados, compararemos con otros algoritmos y con otras investigaciones ya que usan el mismo algoritmo.

La última parte es la **Conclusión** de ahí veremos todas las resolución de este problema y determinaremos si nuestro algoritmo ha cumplido con las expectativas.

3. OBJETIVOS

Los objetivos de esta documentación es explicar los todos los conceptos teóricos relacionados con la optimización Software y las diferentes metodologías usadas para llevarlas a cabo, además indagaremos en una de ellas para demostrar que esto se puede llevar a cabo con resultados reales.

Explicando un poco más, este método será explicado previamente antes de demostrarlo ya que si no, nos sentiríamos perdidos durante toda la documentación.

4. ANTECEDENTES

Esta parte está orientada a entender todo lo referente a teoría, si no nos detenemos en cada apartado, es muy probable que nos sintamos bastantes perdidos en el resto de la documentación.

4.1 Requisitos Softwares

Todo lo que vamos a ver en esta sección está orientado a entender todos los conceptos teóricos que vamos a ver sobre los requisitos softwares.

Antes de entrar en materia y empezar a explicar conceptos, lo que vamos a hacer es dar una breve introducción sobre cómo funcionan los “requisitos softwares”.

Todo proyecto software siempre está basado en plazo de entregas, en saber qué hay que hacer en cada momento y claro, el tiempo del que se dispone es bastante limitado, por consiguiente, debe minimizarse el tiempo empleado en cada tarea y tenerlo todo planificado antes de proceder con el desarrollo de una tarea. También debemos de tener en cuenta de los recursos que disponemos son limitados.

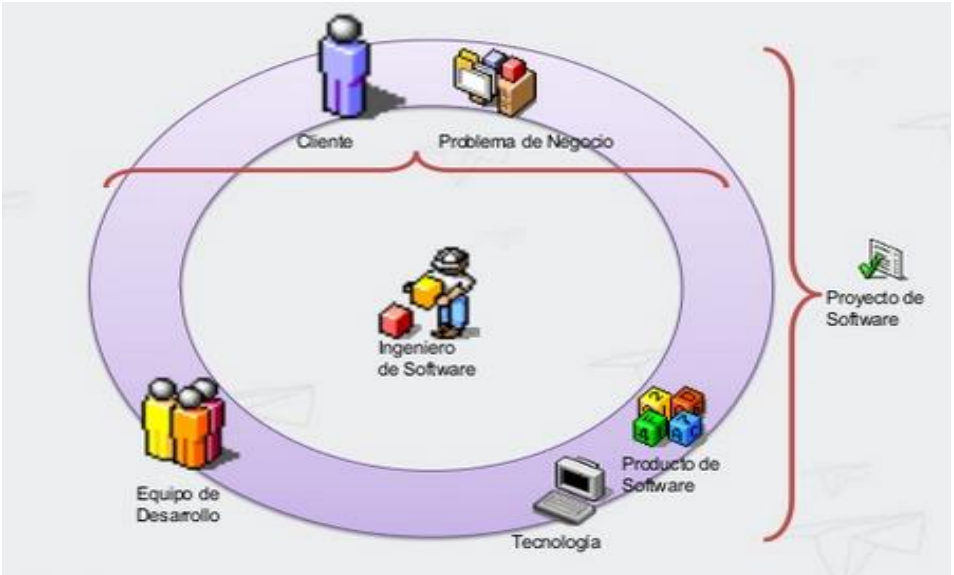


Figura 1: Esquema de organización de un proyecto software, está sacada de https://es.slideshare.net/leansight/qu-metodologa-ser-ms-adecuada-para-mi-proyecto-software-13905273.

Para que un proyecto software salga a delante sin problemas la planificación de cada tarea debe estar bien calculada, es decir, que ha de desarrollarse en el momento planificado. Por supuesto y damos a entender que todo no se puede realizar, ya que cada tarea conlleva un tiempo y un esfuerzo realizarse.

Este proyecto se encarga de seleccionar los diferentes requisitos que se van a incluir en la siguiente entrega del proyecto ya como he explicado en el párrafo anterior, es complicado cumplir con los plazos de entrega, de hecho, en diversas ocasiones no se pueden desarrollar todos los requisitos planteados, por eso existe la optimización de requisitos sofwares.

La optimización es esencial en la IS (Ingeniería Software) y sobre todo cuando hablamos de desarrollar y/o planificar un proyecto. Existen diversos tipos de optimización, podremos ver algunas a lo largo de esta documentación. Una técnica que vamos a implementar para desarrollar nuestro producto, está basada en metodologías ágiles, es decir, está enfocado a incrementar el software en cada entrega. En definitiva, se van generando entregas sobre el proyecto software a corto plazo, por tanto, se disponen de nuevos requisitos en cada entrega.

La ingeniería software si tiene una complicación es que debe seleccionar estos requisitos en cada entrega, pero hay diferentes prioridades para desarrollar dicho requisito, como el tiempo de costo, la influencia que tiene hacia otro requisito o viceversa,… No es fácil hallar una solución, ya que depende de diversos factores.

4.1.1 NRP (Next Release Problem)

Como hemos dicho en la introducción de los conceptos teóricos, la optimización de requisitos es una ardua tarea, NRP se encarga de esta tarea. NRP debe seleccionar las características para la siguiente entrega, de modo que debe seleccionar las características con menos costo de desarrollo y que satisfagan en lo máximo posible a los clientes.

Nos percatamos de que el problema no consta en que solo no tenemos una característica a evaluar, si no que disponemos de dos característica y ambos, son igual de importantes a la hora de sopesar la selección de una característica.

Hubo varios métodos de selección de atributos antes del nuestro, por eso vamos a explicarlos brevemente antes de entrar más a fondo en materia.

Uno de los métodos se llama: **Proceso Analítico Jerárquico** o PAJ, en inglés sería **AHP** (Analytical Hierarchy Process), donde los requisitos son calificado según un valor-costo.

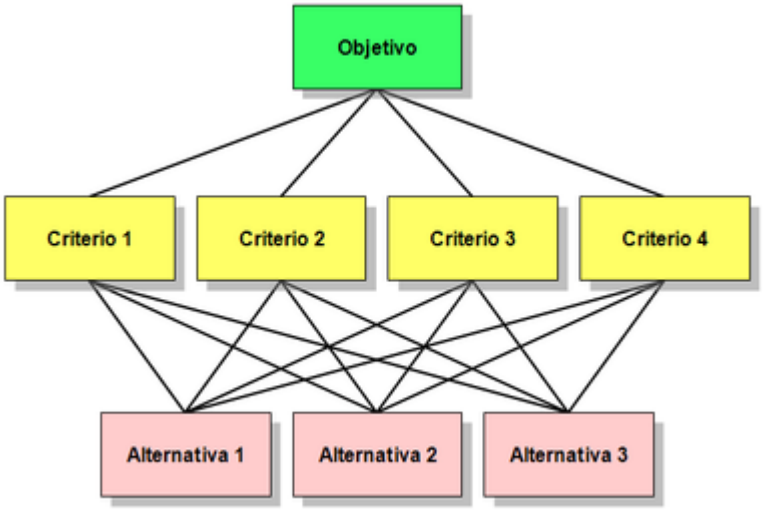


Figura 2: Estructura del proceso analítico Jerárquico. Esta imagen está sacada de https://es.wikipedia.org/wiki/Proceso_anal%C3%Adtico_jer%C3%A1rquico.

Explicando esto un poco mejor, el problema de seleccionar una característica, es complicado ya que hay que darle un peso a cada recurso, por ello se queda en grupo y se debate qué desea cada uno/a incluir en la siguiente entrega, a cada característica se le da un cierto valor, por tanto, cada miembro del grupo, normalmente suelen ser los jefes del proyecto a realizar y por supuesto, cada uno tendrá sus preferencias. En este tipo de reuniones, se vota y se decide, qué prioridad real van a tener cada recurso e intentar llegar a un consenso para poder elegir los recursos de la siguiente entrega, este

caso está más orientado a nuestro proyecto, pero la verdad es que se puede orientar de varias maneras.

Antes de proceder a generalizar, AHP se basa en un sistema de modelo o jugadores(empresarios/jefes/empleados). Si el grupo tiene objetivos significativamente diferentes y no puede reunirse para discutir la decisión, cada miembro del grupo puede emitir un juicio por separado, basándose en modelos o jugadores por separado. Si se basa en modelos separados, cada miembro del grupo ingresa su juicio en un modelo separado, que luego será promediado. Si está basado en jugadores, se establece un modelo combinado de cada jugador, evaluando cada uno de los factores en los que está basado este modelo. Un ejemplo de lo que debería de ser una tabla de valores de cada recurso para poder verlo con mejor claridad.

	A	B	C	D	E	F	G	H	I	J	V	X	Y
1		SR-1	SR-2	SR-3	SR-4	SR-5	SR-6	SR-7	SR-8	SR-9	Scores	Product	Ratio
2	SR-1	1	8	1/5	3	1	2	2	3	1	0.1373	1.5427	11.2344
3	SR-2	1/8	1	1/5	1/7	1/7	1/7	1/7	1/9	1/9	0.0146	0.1549	10.5917
4	SR-3	5	5	1	1	2	1	3	1	1	0.1717	1.9647	11.4415
5	SR-4	1/3	7	1	1	1/2	1/2	3	1/2	1	0.0968	1.0743	11.0955
6	SR-5	1	7	1/2	2	1	3	3	1	1/3	0.1259	1.4065	11.1681
7	SR-6	1/2	7	1	2	1/3	1	1/3	1	1	0.0911	0.9550	10.4813
8	SR-7	1/2	7	1/3	1/3	1/3	3	1	3	2	0.1155	1.2740	11.0301
9	SR-8	1/3	9	1	2	1	1	1/3	1	1/6	0.0887	0.9134	10.2961
10	SR-9	1	9	1	1	3	1	1/2	6	1	0.0887	1.7547	11.0884
11												CI	0.2420
12												CI/RI	0.1669

Figura 3: tabla de valores y prioridades AHP. Esta figura está sacada de https://www.us-cert.gov/bsi/articles/best-practices/requirements-engineering/requirements-prioritization-case-study-using-ahp.

Otro de los métodos era el **Despliegue de la Función de Calidad** o DFC, que en inglés sería **QFD** (Quality Function Deployment), en este método se organizan en una escala, prácticamente traduce los requisitos del usuario en requisitos técnicos del proyecto. Este método, ha sido el promotor del desarrollo de un proyecto mediante entregas.

Para entender este método correctamente, debemos explicarlo con esta matriz:

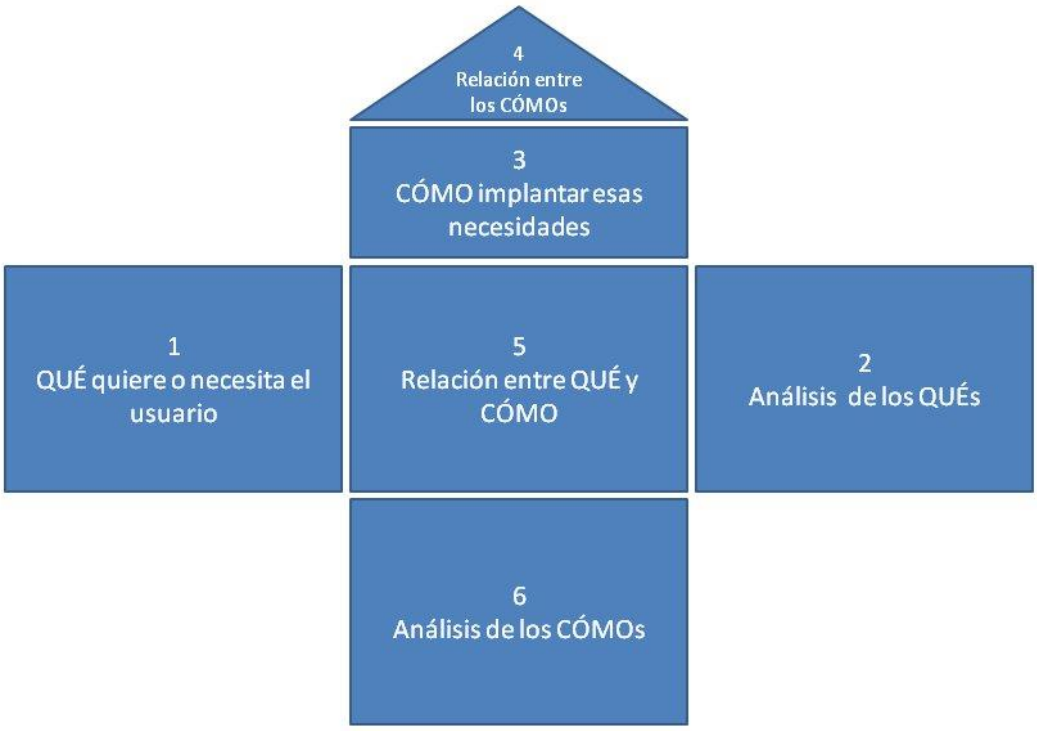


Figura 4: Matriz de QFD, esta imagen está sacada de https://www.pdcahome.com/wp-content/uploads/2012/10/partes-de-un-qfd2.jpeg.

- **La sección horizontal** se encarga de saber qué, cómo y cuantos requisitos quiere el cliente y determina la importancia de estos requisitos.

- **La sección vertical** se encarga de obtener la información relevante a cerca de los clientes. Interpreta los requisitos de los clientes de manera que pueda acotarlos, es decir, darle una medida y examina la relación que hay entre este requisito y el cliente, además, también nos da la información acerca de las metas fijadas por el cliente, sus datos técnicos.

- **La sección central** se encarga de especificar el nivel de funcionamiento que ha de ser alcanzado, así como cuidar la interrelación que hay entre los requisitos. El objetivo real de esta sección se encarga de resolver cualquier conflicto que haya entre los requisitos.

Pero hay un problema con estos dos métodos y es que ambos no son compatibles con todas las dependencias que existen entre los requisitos. Estos métodos no se percatan de que estos recursos son necesidades reales actualmente y que acarrean muchas comparaciones cuando la escala del proyecto empieza a aumentar.

Pero todo lo dicho en el anterior párrafo, no indica que no nos sirvan los anteriores métodos, sino todo lo contrario. Nuestro TFG está basado en estos dos métodos, conceptual mente toma las bases de estos dos métodos para poder resolver el problema multiobjetivo, pero primero debemos explicar la optimización software.

Cuando se halló la dificultad de la selección de requisitos en la Ingeniería Software, se fundamentó como un problema de mono-objetivo en un campo de Ingeniería de Software Basada en Búsqueda ISBB o SBSE (Serach Based in Software Engenieer).

SBSE es el campo de que se encarga en la optimización de los algoritmos en búsquedas que abordan problemas de Ingeniería Software. A lo largo de los últimos años, estos problemas de ingeniería, se han resuelto con diferentes métodos metaheurísticos, pero no se le daba el enfoque adecuado, ya que solo se centraban en un solo objetivo. Es fácil aplicar estos métodos cuando solo debemos cubrir un objetivo, pero cuando los requisitos interactúan entre sí, no se pueden aplicar estos métodos. Desarrollar estos problemas multi-objetivos con un solo objetivo, para poder aplicar estos métodos tiene un inconveniente, y es que cuando desarrollamos este proyecto con un solo objetivo, el resultado que nos da está acotado, ya que orientamos el proyecto hacia un solo punto, por ese motivo, necesitamos un desarrollo multi-objetivo

Prácticamente el método de NRP ha sido planteado recientemente como un problema de optimización de multi-objetivo (MOOP - multi-objective optimization problem). Cuando se planteó por primera vez el problema de multiobjetivo para NRP (MONRP), esta se consideraba sin tener en cuenta la

relación que había entre cada requisito, sin límite de costo y también, teniendo en cuenta que cada objetivo se encaraba por separado, es decir, que no se relacionaba nada con nada, como he explicado hace poco, no existía interrelación entre recursos y tampoco interrelación entre objetivos.

Cuando se veía que no existía interrelación en requisitos u objetivos, se propuso una primera solución, la cual consiste en la optimización a la hora de hacerle caso a un cliente, es decir, se priorizaban los conflictos que existían entre las prioridades de los clientes. Esto en primera instancia, estamos resolviendo el problema de los objetivos, pero se deja de lado el problema de la selección de requisitos. Para poder resolver este problema, se propusieron algoritmos evolutivos en inspiración cuántica como: PAES (Pareto Archived Evolution Strategy – Estrategia de Evolución Archivada de Pareto), NSGA-II (Fast Non-dominated Sorting Genetic Algorithm - Algoritmo genético de ordenación no dominado rápido) y MOCell (MultiObjective Cellular Genetic Algorithm - MultiObjetivo Algoritmo Genético Celular). Pero todos estos algoritmos siguen teniendo un problema y es que no resuelven del todo bien la interrelación entre los recursos, al final se resolvió con un algoritmo de Optimización por Colonia de Hormiga (ACO - Ant Optimization Colony), hablaremos de estos algoritmos en el documento más adelante para que entendamos todo con mayor claridad.

4.2 Técnicas de optimización

Para hablar de las técnicas de optimización, debemos primero hablar sobre qué métodos hay para hallar una solución del problema, para ello tenemos los métodos exactos y los métodos aproximados, nos centraremos en los métodos aproximados, lo veremos a continuación. En general esta información ha sido sacada de *Álvaro García Sánchez.*

Las técnicas heurísticas son un conjunto de pasos que han de realizarse en el menor tiempo posible para hallar una solución de un determinado problema. El enfoque del problema usando determinadas reglas ocasionará una solución buena o muy cercana a lo que deseamos. El método heurístico es un procedimiento para resolver un problema de optimización mediante una aproximación intuitiva, para obtener una buena solución.

Por otro lado si el método heurístico lo centramos en metodologías exactas, las heurísticas se centraran en encontrar una solución siguiendo ciertos criterios, esta solución hallada será óptima pero no la más óptima que se pueda encontrar, esto sería una contra, pero uno de los beneficios que tiene aplicar heurísticas a métodos exactos es que el tiempo de ejecución es más corto.

Normalmente las heurísticas se usan cuando el método no podemos ofrecer una solución exacta del problema o sí que existe, pero el método encargado de ello, requiere de demasiado tiempo para hallar una solución. Para acortar ese tiempo y encontrar un resultado bastante óptimo, usamos los métodos heurísticos.

Los métodos heurísticos se pueden clasificar en diferentes categorías dependiendo de su modo de resolución del problema, nombremos algunos de ellos a continuación:

Métodos de descomposición: son métodos centrados en descomponer el problema en subproblemas más sencillos de resolver.

- **Métodos inductivos:** pretenden generalizar de versiones pequeñas al caso completo.

- **Métodos de búsqueda local:** en las que se parte de una solución inicial a la que se realizan modificaciones en sucesivas iteraciones para obtener una solución final. En cada iteración existe un conjunto de soluciones vecinas candidatas a ser nueva solución en el proceso. En este grupo se encuadran las técnicas metaheurísticas y en lo que nos vamos a basar a la hora de desarrollar nuestro problema.(Estos son los llamados algoritmos genéticos)

- **Métodos constructivos:** son deterministas y consisten en construir paso a paso una solución del problema, y suelen mejorar la elección en cada iteración.

- **Métodos de manipulación de modelo:** obtienen una solución del problema original a partir de otra de otro problema simplificado (con menos restricciones, linealizando el problema, etc.)

4.2.1 Metaheurísticas

Nuestro problema debe aplicarse a las Metaheurísticas, estas sirven para resolver un tipo de problema computacional general, usando los parámetros dados por el usuario sobre unos procedimientos genéricos y abstractos de una manera que se espera eficientemente. Normalmente, estos procedimientos son heurísticos que ya han sido explicados previamentes. El nombre combina el prefijo griego "meta" ("más allá", aquí con el sentido de "nivel superior") y "heurístico" (*heuriskein*, "encontrar").

Las metaheurísticas son procesos que no garantizan encontrar la solución más óptima, ya que están basadas en reglas relativamente sencillas. La diferencia que existe contra los métodos heurísticos, es que tratan de huir de

los óptimos locales orientando la búsqueda dependiendo de la evolución que vaya teniendo el algoritmo de búsqueda.

Estos métodos están basados en la optimización combinatoria, es decir, son problemas en los que la variable de decisión son enteras en las que, generalmente, el espacio de soluciones está formado por ordenaciones de valores de dichas variables, sin embargo, las metaheurísticas también se pueden aplicar a problemas con variables continuas.

Las técnicas metaheurísticas se semejan al disponer un punto de partida de una solución, es decir, una solución. Esta solución no necesariamente tiene que ser óptima, pero a partir de ella, se puede ir obteniendo diversas soluciones, por supuesto, está irá cambiando según ciertos requisitos, esto es la evolución. No debe ser necesariamente una única solución que vaya evolucionando, puede ser un conjunto de soluciones (población) que vayan evolucionando.

Las técnicas metaheurísticas más conocidas son: los algoritmos genéticos, la búsqueda tabú, el recocido simulado, la búsqueda “scatter”, las colonias de hormigas, las ranas saltarinas, las redes neuronales, también incluidas entre las técnicas metaheurísticas.

Todas las metaheurísticas tienen las mismas especificaciones:

- Son ciegas, no saben si llegan a la solución óptima. Por lo tanto, se les debe indicar cuando deben acabar.

- Son algoritmos aproximativos y, por lo tanto, no garantizan la obtención de la solución óptima. Aceptan ocasionalmente malos movimientos (es decir, se trata de procesos de búsqueda en los que cada nueva solución no es necesariamente mejor –en términos de la función objetivo– que la inmediatamente anterior). Algunas veces aceptan, incluso, soluciones no factibles como paso intermedio para acceder a nuevas regiones no exploradas.

- Son relativamente sencillos; todo lo que se necesita es una representación adecuada del espacio de soluciones, una solución inicial (o un conjunto de ellas) y un mecanismo para explorar el campo de soluciones.

- Son generales. Prácticamente se pueden aplicar en la resolución de cualquier problema de optimización de carácter combinatorio. Sin embargo, la definición de la técnica será más o menos eficiente en la medida en que las operaciones tengan relación con el problema considerado.

- La regla de selección depende del instante del proceso y de la historia hasta ese momento. Si en dos iteraciones determinadas, la solución es la misma, la nueva solución de la siguiente iteración no tiene por qué ser necesariamente la misma, en general, no lo será.

También podemos clasificar las metaheurísticas en dos tipos, las primeras son las metaheurísticas basadas en **trayectorias** y el otro tipo de metaheurísticas son las metaheurísticas basadas en **población**. Nosotros nos centraremos en las metaheurísticas basadas en **población**.

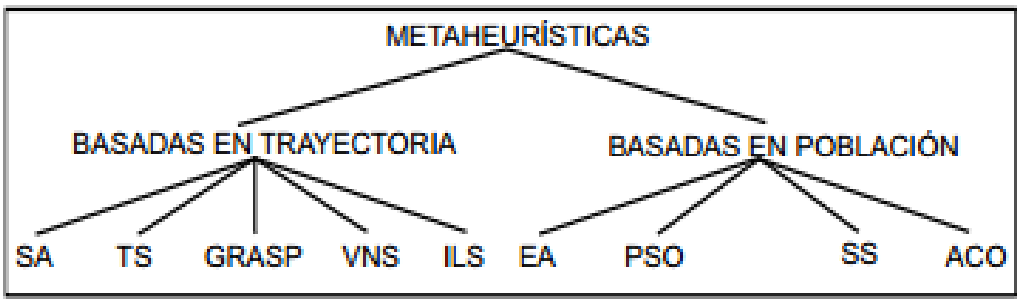


Figura 5: Clasificación de las metaheurísticas. Imagen obtenida de José Manuel García Nieto, Enrique Alba Torres y Gabriel Jesús Luque Polo – 2006 – Algoritmos Basados en Cúmulos de Partículas Para la Resolución de Problemas Complejos.

Expliquemos brevemente en qué se diferencian ambas técnicas metaheurísticas:

- **Trayectoria:** La principal característica de estos métodos es que parten de un punto y mediante la exploración del vecindario van actualizando la solución actual, formando una trayectoria. Las principales técnicas son: El Enfriamiento Simulado o Simulated Annealing (SA), La Búsqueda Tabú o Tabu Search (TS), El Procedimiento de Búsqueda Miope Aleatorizado y Adaptativo o The Greedy Randomized Adaptive Search Procedure (GRASP), La Búsqueda en Vecindario Variable o Variable Neighborhood Search (VNS) y La Búsqueda Local Iterada o Iterated Local Search (ILS).

Hablaremos sobre las técnicas basadas en población en el siguiente apartado.

4.3 Algoritmos basados en población

Estos algoritmos utilizan alguna clase de estructuración de los individuos de la población, dicha información ha sido sustraida de *José Francisco Chicano García – 2007*. Este esquema es ampliamente utilizado especialmente en el campo de los algoritmos evolutivos, en el cual nos centraremos por ser nuestra propuesta al problema que estamos abordando durante la documentación. Entre los esquemas más populares para estructurar la población encontramos el modelo distribuido (o de grano grueso) y el modelo celular (o de grano fino).

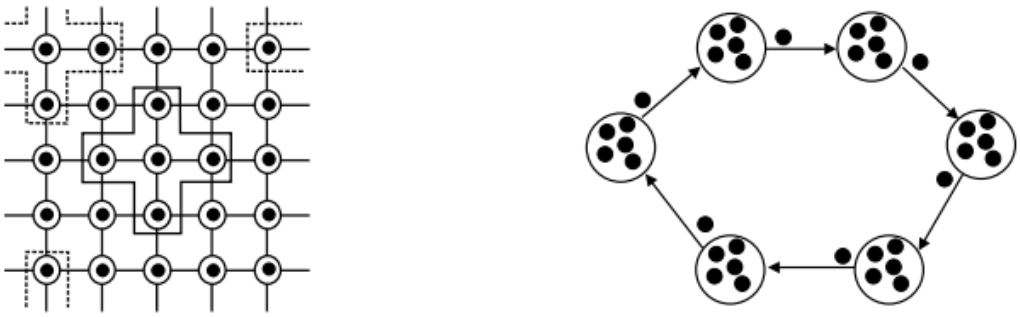


Figura 6: Los dos modelos más populares para estructurar la población, a la izquierda el modelo celular y a la derecha el modelo distribuido. Esta imagen ha sido obtenida de José Francisco Chicano García – 2007 – Metaheurísticas e Ingeniería del Software.

Por un lado, las metaheurísticas celulares (véase el esquema de la izquierda en la Figura 6) se basan en el concepto de vecindario1. Cada individuo tiene a su alrededor un conjunto de individuos vecinos donde se lleva a cabo la explotación de las soluciones. La exploración y la difusión de las soluciones al resto de la población se producen debido a que los vecindarios están solapados, lo que produce que las buenas soluciones se extiendan lentamente por toda la población.

Por otro lado, en el caso de los algoritmos distribuidos (véase el esquema de la derecha en la Figura 6), la población se divide entre un conjunto de islas que ejecutan una metaheurística secuencial. Las islas cooperan entre sí mediante el intercambio de información (generalmente individuos, aunque nada impide intercambiar otro tipo de información). Esta cooperación permite

introducir diversidad en las subpoblaciones, evitando caer así en los óptimos locales.

Para terminar de definir este esquema el usuario debe dar una serie de parámetros como: topología, que indica a dónde se envían los individuos de cada isla y de dónde se pueden recibir; periodo de migración, que es el número de iteraciones entre dos intercambios de información; tasa de migración, que es el número de individuos emigrados; criterio de selección de los individuos a migrar y criterio de reemplazo, que indica si se reemplazan algunos individuos de la población actual para introducir a los inmigrantes y determina qué individuos se reemplazarán. Finalmente, se debe decidir si estos intercambios se realizan de forma síncrona o asíncrona.

4.3.1 Tipos de Algoritmos basados en población

Existen varios tipos de algoritmos basados en población generalmente este apartado ha sido sustraído de *José Manuel García Nieto, Enrique Alba Torres y Gabriel Jesús Luque Polo – 2006*. El principal algoritmo en este apartado es el primero, ya que a raíz de ese, saldrán varios algoritmos, estos algoritmos, son variantes de este mismo, pero son igual de importantes.

Los **algoritmos evolutivos** o **Evolutionary Algorithms** *(EA)* están inspirados en la teoría de la evolución natural. Esta familia de técnicas sigue un proceso iterativo y estocástico que opera sobre una población de soluciones, denominadas en este contexto individuos. Inicialmente, la población es generada aleatoriamente (quizás con ayuda de un heurístico de construcción). El esquema general de un algoritmo evolutivo comprende tres fases principales: selección, reproducción y reemplazo. El proceso completo es repetido hasta que se cumpla un cierto criterio de terminación (normalmente después de un número dado de iteraciones).

Los **algoritmos de estimación de la distribución** o **Estimation of Distribution Algorithms** *(EDA)* muestran un comportamiento similar a los algoritmos evolutivos presentados en la sección anterior y, de hecho, muchos

autores consideran los EDA como otro tipo de EA. Los EDA operan sobre una población de soluciones tentativas como los algoritmos evolutivos pero, a diferencia de estos últimos, que utilizan operadores de recombinación y mutación para mejorar las soluciones, los EDA infieren la distribución de probabilidad del conjunto seleccionado y, a partir de esta, generan nuevas soluciones que formarán parte de la población.

La **búsqueda dispersa** o **Scatter Search** *(SS)* es una metaheurística cuyos principios fueron presentados en y que actualmente está recibiendo una gran atención por parte de la comunidad científica. El algoritmo mantiene un conjunto relativamente pequeño de soluciones tentativas (llamado conjunto de referencia o RefSet) que se caracteriza por contener soluciones de calidad y diversas (distantes en el espacio de búsqueda). Para la definición completa de SS hay que concretar cinco componentes: creación
de la población inicial, generación del conjunto de referencia, generación de subconjuntos de soluciones, método de combinación de soluciones y método de mejora.

Los algoritmos de optimización basados en **colonias de hormigas** o **Ant Colony Optimization** *(ACO)* están inspirados en el comportamiento de las hormigas reales cuando buscan comida. Este comportamiento es el siguiente: inicialmente, las hormigas exploran el área cercana a su nido de forma aleatoria. Tan pronto como una hormiga encuentra comida, la lleva al nido. Mientras que realiza este camino, la hormiga va depositando una sustancia química denominada feromona. Esta sustancia ayudará al resto de las hormigas a encontrar la comida. La comunicación indirecta entre las hormigas mediante el rastro de feromona las capacita para encontrar el camino más corto entre el nido y la comida. Este comportamiento es el que intenta simular este método para resolver problemas de optimización.

Los algoritmos de **optimización basados en cúmulos de partículas** o **Particle Swarm Optimization** (PSO) están inspirados en el comportamiento social del vuelo de las bandadas de aves o el movimiento de los bancos de peces. El algoritmo PSO mantiene un conjunto de soluciones, también

llamadas partículas, que son inicializadas aleatoriamente en el espacio de búsqueda. Cada partícula posee una posición y velocidad que cambia conforme avanza la búsqueda. En el movimiento de una partícula influye su velocidad y las posiciones donde la propia partícula y las partículas de su vecindario encontraron buenas soluciones.

4.4 Algoritmos evolutivos

Alrededor de los años 60, algunos investigadores visionarios coincidieron (de forma independiente) en la idea de implementar algoritmos basados en el modelo de evolución orgánica como un intento de resolver tareas complejas de optimización en ordenadores dicha información ha sido sustraída de *José Francisco Chicano García – 2007*. Hoy en día, debido a su robustez, a su amplia aplicabilidad, y también a la disponibilidad de una cada vez mayor potencia computacional, e incluso programas paralelos, el campo de investigación resultante, el de la computación evolutiva, recibe una atención creciente por parte de los investigadores de un gran número de disciplinas. El marco de la computación evolutiva establece una aproximación para resolver el problema de buscar valores óptimos mediante el uso de modelos computacionales basados en procesos evolutivos (algoritmos evolutivos). Los EA son técnicas de optimización que trabajan sobre poblaciones de soluciones y que están diseñadas para buscar valores óptimos en espacios complejos. Están basados en procesos biológicos que se pueden apreciar en la naturaleza, como la selección natural o la herencia genética. Parte de la evolución está determinada por la selección natural de individuos diferentes compitiendo por recursos en su entorno. Algunos individuos son mejores que otros y es deseable que aquellos individuos que son mejores sobrevivan y propaguen su material genético.

La reproducción sexual permite el intercambio del material genético de los cromosomas, produciendo asé descendientes que contienen una combinación de la información genética de sus padres. Este es el operador de recombinación utilizado en los EA, también llamado operador de cruce. La recombinación ocurre en un entorno en el que la selección de los individuos que tienen que emparejarse depende, principalmente, del valor de la función de fltness del individuo, es decir, de cómo de bueno es el individuo comparado con los de su entorno.

Como en el caso biológico, los individuos pueden sufrir mutaciones ocasionalmente (operador de mutación). La mutación es una fuente

importante de diversidad para los EA. En un EA, se introduce normalmente una gran cantidad de diversidad al comienzo del algoritmo mediante la generación de una población de individuos aleatorios. La importancia de la mutación, que introduce aún más diversidad mientras el algoritmo se ejecuta, es objeto de debate. Algunos se refieren a la mutación como un operador de segundo plano, que simplemente reemplaza parte de la diversidad original que se haya podido perder a lo largo de la evolución, mientras que otros ven la mutación como el operador que juega el papel principal en el proceso evolutivo.

En la Figura 7 se muestra el esquema de funcionamiento de un EA típico. Como puede verse, un EA procede de forma iterativa mediante la evolución de los individuos pertenecientes a la población actual. Esta evolución es normalmente consecuencia de la aplicación de operadores estocásticos de variación sobre la población, como la selección, recombinación y mutación, con el fin de calcular una generación completa de nuevos individuos. El criterio de terminación consiste normalmente en alcanzar un número máximo de iteraciones (programado previamente) del algoritmo, o encontrar la solución óptima al problema (o una aproximación a la misma) en caso de que se conozca de antemano.

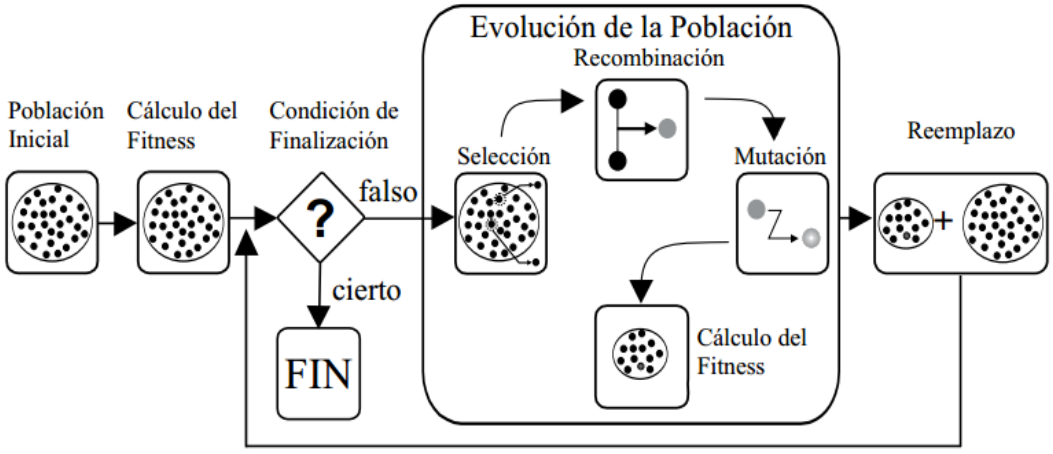


Figura 7: Funcionamiento de un EA canónico. Imagen sacada de José Francisco Chicano García – 2007 – Metaheurísticas e Ingeniería del Software.

Pasaremos a explicar el proceso evolutivo de la población (Figura 7) el cual consta de tres partes: selección, reproducción y reemplazo. A continuación detallamos estas tres fases:

- **Selección:** partiendo de la población inicial P de μ individuos, se crea una nueva población temporal (P0) de λ individuos. Generalmente los individuos más aptos (aquellos correspondientes a las mejores soluciones contenidas en la población) tienen un mayor número de instancias que aquellos que tienen menos aptitud (selección natural). De acuerdo con los valores de μ y λ podemos definir distintos esquemas de selección.

- **Reproducción:** en esta fase se aplican los operadores reproductivos a los individuos de la población P0 para producir una nueva población. Típicamente, esos operadores se corresponden con la recombinación de parejas y con la mutación de los nuevos individuos generados. Estos operadores de variación son, en general, no deterministas, es decir, no siempre se tienen que aplicar a todos los individuos y en todas las generaciones del algoritmo, sino que su comportamiento viene determinado por su probabilidad asociada.

- **Reemplazo:** finalmente, los individuos de la población original son sustituidos por los individuos recién creados. Este reemplazo usualmente intenta mantener los mejores individuos eliminando los peores.

Estos algoritmos establecen un equilibrio entre la explotación de buenas soluciones (fase de selección) y la exploración de nuevas zonas del espacio de búsqueda (fase de reproducción), basado en el hecho de que la política de reemplazo permite la aceptación de nuevas soluciones que no mejoran necesariamente las existentes.

4.4.1 Técnicas evolutivas

Fueron cuatro los primeros tipos de algoritmos evolutivos que surgieron esta información se ha obtenido de *José Francisco Chicano García – 2007*. Estas cuatro familias de algoritmos fueron desarrolladas:

- **Algoritmos genéticos** (*GA – Genetic Algorithms*)**:** Por un lado, estos patrones permiten que con el transcurso del tiempo se exploren continuamente nuevas posibilidades y, por otro, y en condiciones normales, raramente conducen a la obtención de individuos absolutamente desadaptados e incapaces de sobrevivir.

Partiendo de una población inicial, es decir, un conjunto inicial de soluciones, se realizan manipulaciones por las que se obtienen sucesivas poblaciones. La función de adaptación indica la bondad de las soluciones consideradas en cada momento.

En cada iteración se realizan una serie de operaciones con los individuos de la población, de entre las cuales las más comunes son: la selección, el cruce, la mutación y la inversión. La aplicación de los operadores anteriores permite obtener, típicamente, soluciones con mejores funciones de adaptación.

Los algoritmos genéticos pertenecen al grupo de las técnicas evolucionarias, que son aquellas técnicas que en cada iteración disponen de un conjunto de soluciones a partir de las cuales obtienen un nuevo conjunto de soluciones.

- **Estrategias evolutivas** *(ES - Evolutionary strategies)***:** cada individuo está formado por tres componentes: las variables del problema (x), un vector de desviaciones estándar (σ) y, opcionalmente, un vector de ángulos (!). Estos dos últimos vectores se usan como parámetros para el principal operador de la técnica: la mutación gaussiana. Los vectores de desviaciones y de ángulos evolucionan

junto con las variables del problema, permitiendo, de este modo, que el algoritmo se autoadapte conforme avanza la búsqueda.

A diferencia de los GA, en las ES el operador de recombinación tiene un papel secundario. De hecho, en la versión original de ES no existía dicho operador. Fue más tarde cuando se introdujo la recombinación en las ES y se propusieron varias alternativas. Además, cada componente de un individuo puede utilizar un mecanismo diferente de recombinación, dando lugar a una gran cantidad de posibilidades para la recombinación de individuos.

- **Programación evolutiva** *(EP - Evolutionary Programming)***:** es prácticamente una variación de los algoritmos genéticos, donde lo que cambia es la representación de los individuos. En el caso de la PE los individuos son ternas (tripletas) cuyos valores representan estados de un autómata finito. Cada terna está formada por: El valor del estado actual, un símbolo del alfabeto utilizado y el valor del nuevo estado.

Estos valores se utilizan, como en un autómata finito, de la siguiente manera: Teniendo el valor del estado actual en el que nos encontramos, tomamos el valor del símbolo actual y si es el símbolo de nuestra terna, nos debemos mover al nuevo estado.

Básicamente así funciona y así se representan los individuos en la PE. Evidentemente las funciones de selección, Cruce (crossover) y mutación deben variar para adaptarse y funcionar con una población de individuos de este tipo.

- **Programación genética** *(GP - Genetic Programming)***:** a es una metodología basada en los algoritmos evolutivos e inspirada en la evolución biológica para desarrollar automáticamente programas de computadoras que realicen una tarea definida por el usuario. Es una especialización de los algoritmos genéticos donde cada individuo es un programa de computadora. Es una técnica de aprendizaje

automático utilizada para optimizar una población de programas de acuerdo a una función de ajuste o aptitud que evalúa la capacidad de cada programa para llevar a cabo la tarea en cuestión.

4.4.2 Tipos de algoritmos evolutivos

Diferentes comportamientos colectivos en la naturaleza véase como el comportamiento de las hormigas de un enjambre o el funcionamiento de una colonia de abejas todo este contenido la hemos obtenido de David L. González – Álvarez – 2013. Estos algoritmos se pueden explicar en algoritmos para poder entender cómo funcionan estos comportamientos. Todos estos comportamientos, se apoyan en las explicaciones de este apartado, veremos algunos ejemplos usados para entender mejor estas técnicas.

La idea básica de la **Optimización por Colonia de Hormigas** (Ant Colony Optimization – ACO): es imitar el comportamiento cooperativo de las hormigas para resolver un problema de optimización. Esta metaheurística puede ser vista como un sistema multiagente donde cada uno de estos agentes está, a su vez, inspirado por el comportamiento de una hormiga. El algoritmo ACO ha sido aplicado para resolver problemas de optimización combinatoria y ha logrado buenos resultados en una gran cantidad de ellos (por ejemplo, problemas de planificación, de enrutado o de asignación).

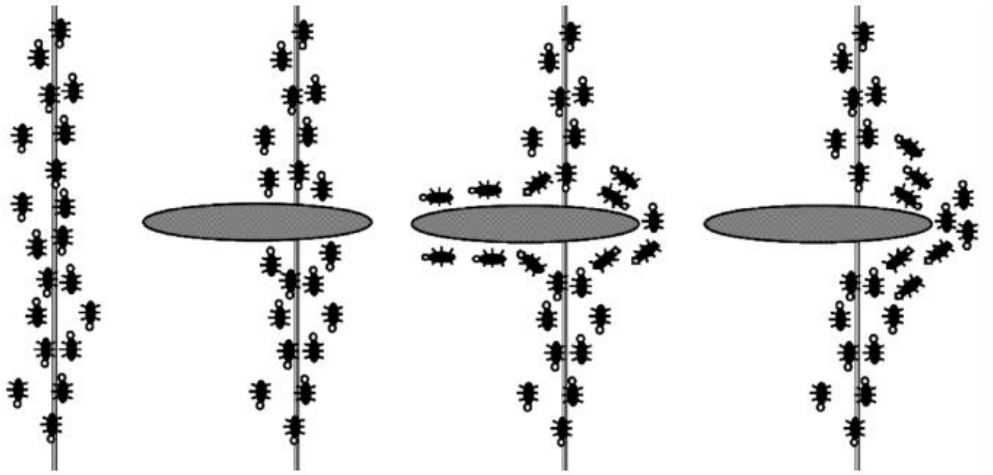


Figura 8: Funcionamiento de un enjambre de hormigas. Imagen sacada de David L. González – Álvarez – 2013 – Metaheurísticas, Optimización Multiobjetivo y Paralelismo para Descubrir Motifs en Secuencias de ADN.

Algunos aspectos interesantes del comportamiento de las hormigas es que simples hormigas, siguiendo un comportamiento colectivo, son capaces de realizar tareas complejas como transportar alimento o encontrar las rutas más cortas entre las fuentes de alimento y sus colonias. Los ACO definen mecanismos de comunicación simples y una hormiga es capaz de encontrar la menor ruta entre dos puntos.

La **Optimización de Enjambre de Partículas** (Particle Swarm Optimization – PSO): es otra metaheurística basada en población inspirada en el comportamiento colectivo. Este algoritmo imita el comportamiento social de organismos naturales como los pájaros o los peces a la hora de encontrar un lugar con el alimento suficiente. Inicialmente, el algoritmo PSO fue diseñado para optimizar problemas continuos. Su primera aplicación en la resolución de un problema de optimización se realizó en el siguiente trabajo.

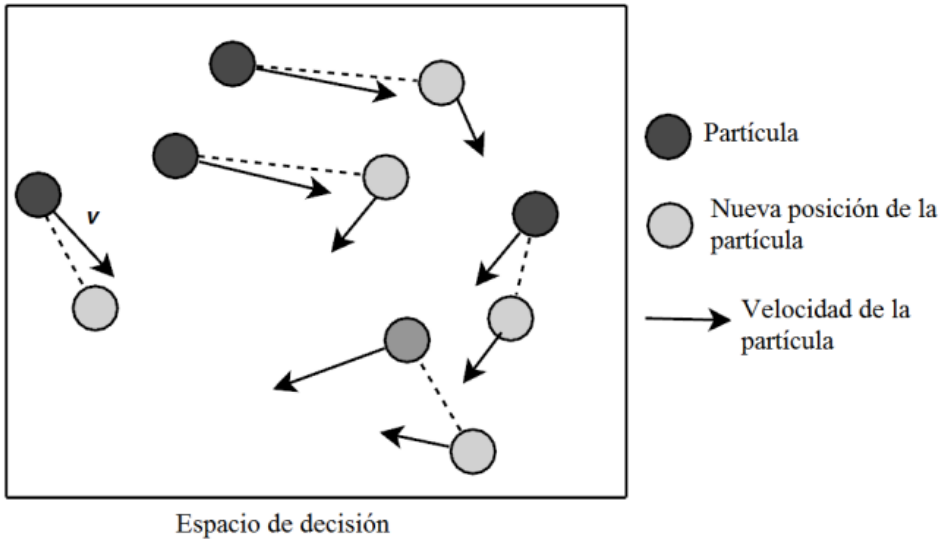


Figura 9: Enjambre de partículas con sus respectivas posiciones y velocidades. Sacada del David L. González – Álvarez – 2013 – Metaheurísticas, Optimización Multiobjetivo y Paralelismo para Descubrir Motifs en Secuencias de ADN.

La **Colonia Artificial de Abejas** (Artificial Bee Colony – ABC): es un algoritmo inspirado por el comportamiento de las abejas de la miel. El algoritmo ABC es un algoritmo basado en población donde los individuos (denominados fuentes de alimento) son explotados por diferentes agentes (distintas clases de abejas). De este modo, el objetivo principal del algoritmo es descubrir las fuentes de alimento con una mayor cantidad de néctar, la cual representa la calidad de la solución.

En el ABC las abejas vuelan alrededor de un espacio de búsqueda multidimensional. Estas abejas presentan diferentes comportamientos, mientras que algunas (abejas obreras y observadoras) escogen fuentes de alimento guiándose por su experiencia y la de la colmena, otras (abejas exploradoras) escogen fuentes de alimento al azar.

Al final el algoritmo irá almacenando las soluciones que presentan una mayor cantidad de néctar. De este modo, el ABC combina procesos de búsqueda local (obreras y observadoras) con procesos de búsqueda global (exploradoras) tratando de balancear las propiedades de explotación y exploración.

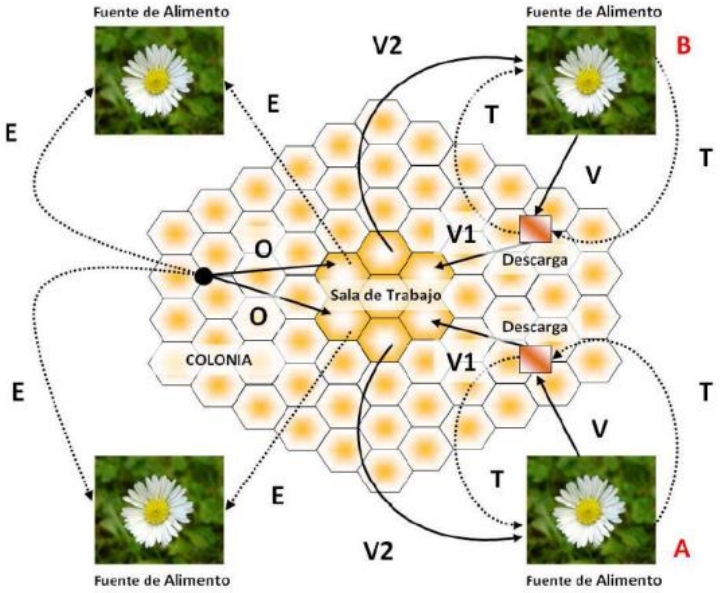


Figura 10: Representación gráfica del funcionamiento del algoritmo MOABC. Imagen obtenida del David L. González – Álvarez – 2013 – Metaheurísticas, Optimización Multiobjetivo y Paralelismo para Descubrir Motifs en Secuencias de ADN.

El **Algoritmo de las Luciérnagas** (Firefly Algorithm – FA): estos algoritmos está inspirado en el comportamiento de las luciérnagas de la luz. Estos insectos pertenecen a una familia de coleópteros polífagos caracterizados por su capacidad de emitir luz (bioluminiscencia). Existen muchas especies y normalmente viven en pantanos o en áreas húmedas y boscosas donde sus larvas pueden alimentarse. Las luciérnagas hembra utilizan su capacidad bioluminiscente para atraer a los machos que vuelan en los alrededores. En este algoritmo hemos asociado estos patrones de luz con los objetivos que deben ser optimizados.

Se formuló este algoritmo teniendo en cuenta las siguientes cuestiones:

- Todas las luciérnagas son asexuales por lo que todas se ven atraídas por todas.

- La atracción es proporcional a su brillo y, dadas dos luciérnagas, la menos brillante se ve atraída por (se mueve hacia) la más brillante, sin embargo, el brillo puede disminuir a medida que la distancia aumenta.

- Si no existen luciérnagas más brillantes que una luciérnaga dada, esta se mueve aleatoriamente.

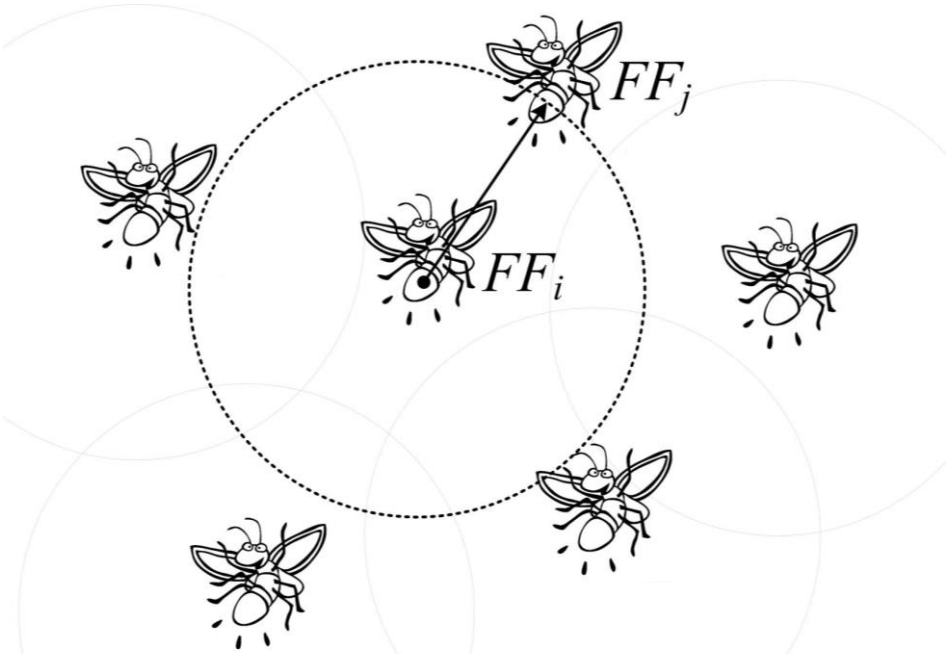


Figura 11: Representación gráfica del algoritmo de las luciérnagas MOFA http://es.mathworks.com/matlabcentral/fileexchange/38931-swarmfirefly-the-firefly-swarm-algorithm--ffsa-?requestedDomain=www.mathworks.com.

El **Algoritmo de Búsqueda Gravitacional** (Gravitational Search Algorithm – GSA): Esta técnica está basada en la ley gravitatoria y la consecuente interacción entre masas. En física, la interacción gravitatoria es una de las cuatro interacciones fundamentales, o lo que es lo mismo, uno de los cuatro tipos de campos cuánticos mediante los cuales interactúan las partículas (las otras son la interacción electromagnética, la interacción nuclear débil y la interacción nuclear fuerte). Cada partícula en el universo se ve atraída por otra. Consecuentemente, la gravedad actúa en todas partes y esto la diferencia de otras fuerzas presentes en la naturaleza.

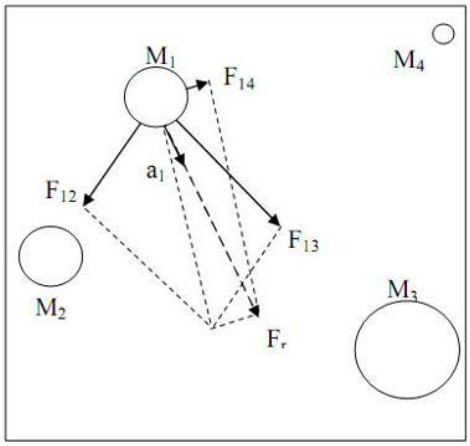


Figura 12: re presentación del algoritmo de búsqueda gravitacional. Imagen obtenida de http://stubartar11.blog.ir/

Nos queda por hablar de un último algoritmo, de hecho es el más importante en este documento, ya que todas las pruebas y resultados se realizan con dicho algoritmo.

4.5 Algoritmo de las ranas

A continuación hablaremos del algoritmo que usaremos esta información la hemos obtenido de *David L. González – Álvarez – 2013*.

El **Algoritmo de las Ranas Saltarinas** (Shuffled Frog Leaping Algorithm - SFLA) es otro algoritmo evolutivo basado en inteligencia colectiva diseñado para abordar problemas de optimización combinatoria. Este algoritmo está inspirado en la evolución de los memes a través de la interacción entre individuos y de un intercambio global de información. El algoritmo SFLA fue diseñado teniendo en cuenta las ventajas que proporcionan los algoritmos genéticos meméticos (MA) y los algoritmos basados en comportamientos sociales como, por ejemplo, el PSO. El concepto memético surge del término meme, el cual simplemente define una unidad de información intelectual o cultural que sobrevive lo suficiente como para ser reconocido como tal y que puede evolucionar a lo largo de generaciones.

Más concretamente, el algoritmo SFLA simula una evolución memética de un conjunto de ranas (soluciones). En este algoritmo las ranas individuales no son importantes ya que son consideradas únicamente como fuentes de información para los diferentes memes. En el SFLA, la población está formada por un conjunto de ranas (soluciones) que se organizan en varios subconjuntos (o memeplexes/charcos), los cuales simulan diferentes especies. En cada memeplex las ranas evolucionan teniendo en cuenta la riqueza genética de los individuos que la componen. Tras un determinado número de pasos evolutivos, algunas ranas son cambiadas de subpoblación para tratar de enriquecer la calidad genética de las ranas de la subpoblación destino. Los procesos de evolución e intercambio se repiten hasta alcanzar la condición de finalización y son los que caracterizan a este algoritmo.

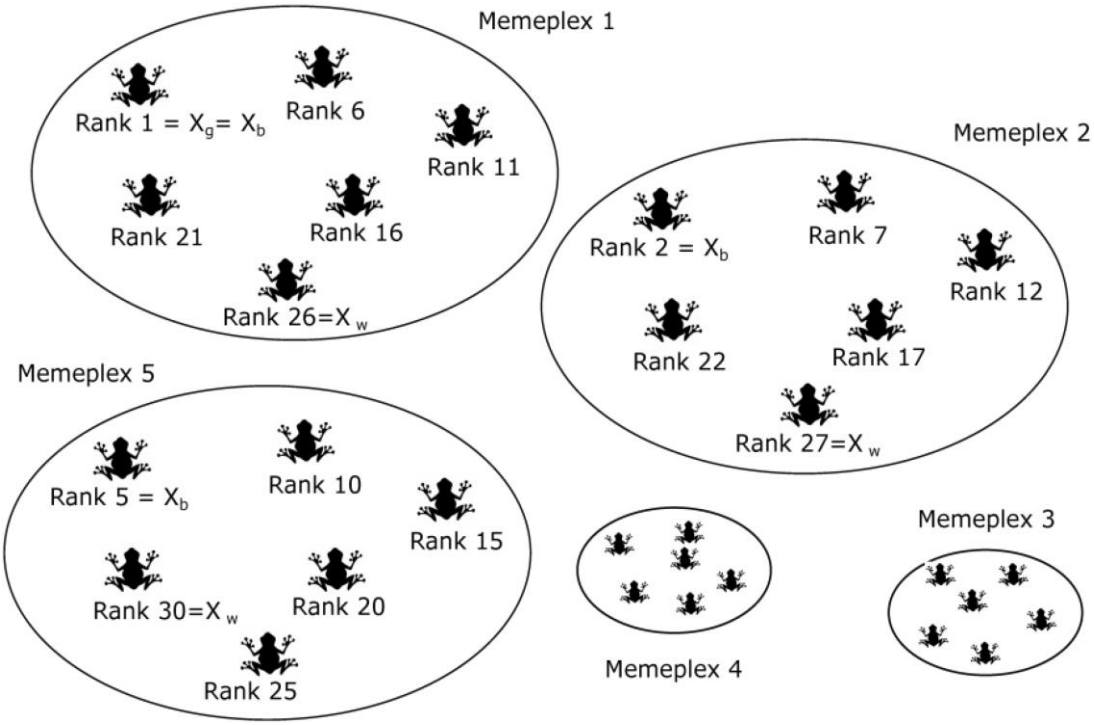


Figura 13: Organización estructural del MO-SFLA. Imagen obtenida de http://www.mdpi.com/2073-4441/8/5/182

4.6 Optimización multiobjetivo

En este apartado como muy bien dice el título, vamos a aprender sobre la optimización multiobjetivo, así como los orígenes de esta, también vamos a hablar sobre cómo ha ido evolucionando a lo largo del tiempo, el cual ha sido poco, ya que la optimización multiobjetivo no tiene mucho tempo. Veremos diversos términos que acompañan a la base de la OMO (optimización multiobjetivo), términos como el frente de pareto, más todas sus características también. Divisaremos otros términos como dominancia, sin el cual este proyecto no se podría realizar y tampoco afianzar bien los conocimientos sobre Pareto. Todos estos términos serán explicados con diferentes fórmulas matemáticas, ya que será necesario explicarlos porque vamos a aplicarlas a nuestro proyecto.

4.6.1 Conceptos previos

Como hemos explicado en el apartado de Requisitos Software (Apartado 4.1) e si se pretendía hacer una OMO (optimización multiobjetivo), simplemente se centraban en un objetivo y el otro no lo tenían en cuenta, o se tenía en cuenta ambos objetivos, pero no se tenía en cuenta la relación que había entre estos. Ya que todo este tema de optimización es relativamente joven y que no solo es aplicable a la optimización software, sino que también se puede extrapolar a muchos otros campos como al punto de vista científico, también si le damos un punto de vista biológico, como muchas otras aplicaciones. Pero todos estos campos tienen varias cosas en común y es que se deben de cumplir las mismas especificaciones, satisfacer lo máximo posible con el menor costo de recursos que sean posibles.

Para que esta optimización se pueda llevar a cabo, primero debemos entender varios conceptos antes de lograr esta optimización y estos conceptos, se apoyan en estudios que a priori, no tienen relación con la optimización software, pero que sin ellos, no se podrían llevar a cabo.

Para empezar a entender los problemas de optimización multiobjetivo (MOP - Multiobjective Optimization Problem), debemos entender y dominar Pareto y el frente de Pareto son dos de las ideas principales. V. Pareto como el nombre bien dice, fue el creador de este concepto. Este concepto fue fundamentado para las ciencias económicas, pero por necesidades, fue integrado en los campos de la ingeniería.

La solución o mejor dicho, soluciones de las que vamos a disponer en MOP, va a ser directamente proporcional al tamaño del problema, por lo tanto no vamos a tener una solución, si no varias soluciones, ya que el tamaño del problema lo exige, por lo tanto, nuestro objetivo va a ser encontrar el conjunto de soluciones más óptima y decidir la que mejor convenga a todos los clientes.

Nuestro objetivo como se ha dicho en el párrafo anterior, va a ser encontrar el conjunto de soluciones más óptimo, pero he aquí el problema de los MOP. Lo complicado de esta cuestión es hallar ese conjunto de soluciones. Ya que disponemos de ese conjunto de soluciones, debemos saber que todas esas soluciones disponen de sus requisitos y que no hay conflictos entre las condiciones de esos requisitos y que, hay relación entre los requisitos y los objetivos. Todo esto será llevado acabo por metaheurísticas (Apartado 4.2.1).

Debemos de tener en cuenta de que en Parteo habrá una solución óptima, y nuestro objetivo será aproximarnos lo máximo posible a esa solución y obtener un conjunto de soluciones lo más cercano posible a esa solución teniendo en cuenta las distintas circunstancias que puedan afectar.

La optimización multiobjetivo se diferencia en varios puntos de la optimización mono-objetivo, no pueden ser iguales, ya que no se orientan de la misma manera al modo de optimización.

A continuación veremos diversos puntos comparando estas dos optimizaciones:

- Si nos fijamos la solución de la optimización multiobjetivo cambia por completo de la mono-objetivo, ya que esta última está optimizada parcialmente si disponemos de varios objetivos a optimizar, por tanto, los resultados serán buenos, pero no todo lo buenos que deseamos.

- El número de soluciones de Pareto óptimas (esto se explicará con más detalle en el siguiente apartado) del que se dispone es mucho mayor, todo siempre teniendo en cuenta el número de objetivos del que se dispongan. Teniendo en cuenta es factor, el número de soluciones puede aumentar de manera exponencial con respecto al tamaño del problema.

- Dependiendo de si el problema es mono-objetivo o multiobjetivo, la forma del frente de pareto cambiará, así como su modalidad o su continuidad. Nos podremos encontrar con un problema con sus soluciones más óptimas en la frontera inferior de una función convexa.

4.6.2 Pareto

Anteriormente ese ha dicho que para entender perfectamente Pareto, primero se ha de dominar completamente los conceptos de “Dominancia de Pareto” y “Frente de Pareto”, a continuación lo vamos a explicar detalladamente para que no haya confusiones y así, podamos entender fácilmente MONRP.

Definición Problema de optimización multiobjetivo. Un problema de optimización multiobjetivo puede definirse como:

Ecuación 1: Definición general de un problema de optimización multiobjetivo.

$$MOP = \begin{Bmatrix} minF(x) = (f_1(x), f_2(x), \ldots, fn(x)) \\ s.c. x \in S \end{Bmatrix}$$

En el marco de la Optimización Multiobjetivo, con frecuencia el decisor razona en términos de la evaluación de una solución según cada objetivo, situándose en el espacio de los objetivos. Para representar el conjunto de soluciones realizables, en el espacio de los objetivos, es preciso determinar la imagen de cada solución realizable del espacio de decisión **(S)**.

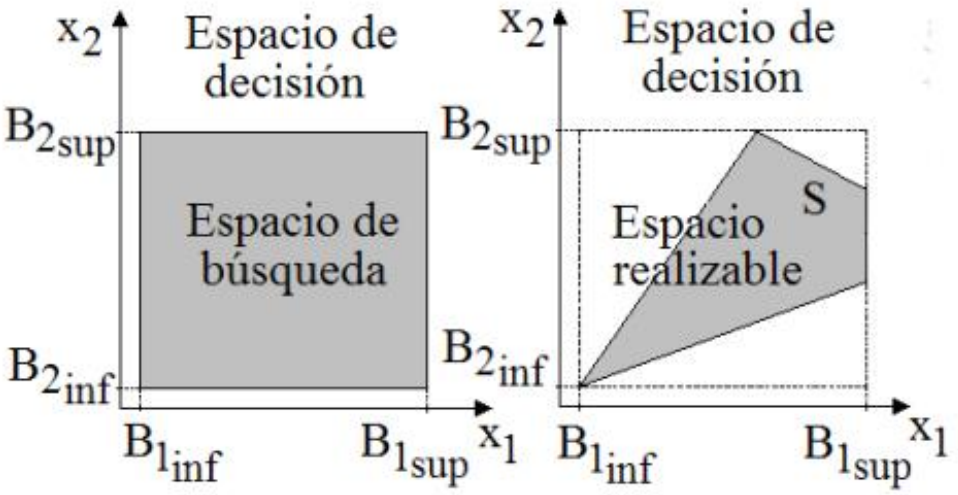


Figura 14: Espacio de búsqueda de soluciones. Imagen sacada de M. Méndez, B. Galván, D. Greiner, G. Winter – Algoritmos Evolutivos y Punto de Funcionamiento Aplicados a un Problema Real de Optimización Multiobjetivo en Diseño de Sistemas de Seguridad.

En Pareto se dispone de un conjunto de soluciones $X = [x1, x2, \ldots, xn]$, donde "n" es el número de objetivos, se dice que domina a$X = [x1, x2, \ldots, xn]$, si Y no es mejor que a X en todos los *i*, esto nos indica de que habrá al menos un Xi que será mejor que su correspondiente Yi. Esto lo que quiere decir, es que ninguna de estas soluciones es dominada entre ellas mismas.

Cuando nos dicen que dos soluciones no están dominadas, cuando ninguna de ellas domina a la otra, para saber si una solución domina a otra, usamos esta fórmula para saber si hay o no dominancia entre resultados y en caso de que haya, se sabrá quién domina a quién.

Dado un vector $\vec{u} = (u1, \ldots, uk)$, se dice que domina a otro vector $X = [x1, x2, \ldots, xn]$ si y sólo si:

Ecuación 2: comparación de dominancia.

$$\forall_i \in \{1, \ldots, k\}, u_i \leq v_i y \exists_{i0} \in (1, \cdots, k) \vee u_{i0} < v_{i0}$$

V. Pareto se apoyó en la definición de dominancia en F. Edgeworth, la definición literaria de dominancia de Pareto:

Una solución Pareto óptima indica que es imposible encontrar una solución que mejore su calidad en algún criterio sin, a su vez, penalizar la calidad de otro criterio.

Definición a Optimalidad de Pareto: Una solución x' se dice que es Pareto-óptima si y sólo si no existe otro vector x tal que $v = f(x) = (v1, \ldots, vk)$ domine a $Y = [y1, y2, \ldots, yn]$.

La definición anterior nos indica que el punto *x'* es un resultado óptimo de Pareto si no existe un vector *x* que haga mejorar alguno de los objetivos, teniendo en cuenta su respectivo x en *x',* sin que empeore de forma simultánea alguno de los otros. La solución a Pareto será un conjunto de soluciones, es decir, será un conjunto de soluciones no dominadas por otras conocidas como con **conjunto de no dominados** o **frente de Pareto**.

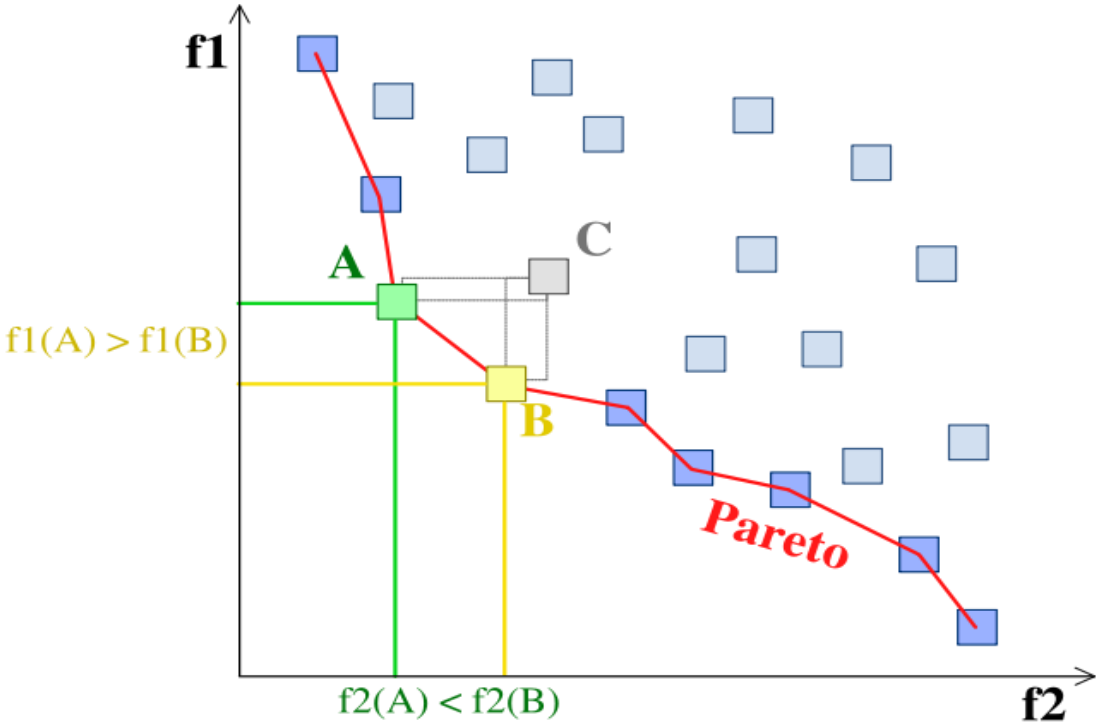


Figura 15: Frente de dominancia de pareto. Imagen obtenida de https://es.wikipedia.org/wiki/Eficiencia_de_Pareto.

Definición Conjunto Pareto óptimo. Para un determinado MOP (F, S), el conjunto, Pareto óptimo se define como sigue:

Ecuación 3: Conjunto de pareto para un determinado problema.

$$P' = \{x \in S / \nexists\, x' \in S, F(x') \prec F(x)\}$$

Definición Frente de Pareto: Para un determinado MOP (F, S) y su correspondiente conjunto Pareto óptimo P', el frente de Pareto se define como:

Ecuación 4: comparación de dominancias de frentes de pareto.

$$PF' = F(x), x \in P'$$

Si nos fijamos, el punto verde (A) no domina al punto amarillo (B), ya que A es mejor que B en un objetivo (f1), pero sin embargo, B es mejor que A en

otro objetivo (f2), por eso no se dominan entre ellas, pero si nos fijamos, si dominan al puto gris (C). Para concluir la explicación de la Figura 16. Si nos fijamos en la línea roja, está el conjunto de soluciones no dominadas o Frente de Pareto, pueden haber más soluciones a parte del frente, pero estás serán dominadas, por tanto no se podrán mostrar en el resultado, esto nos indica que no solo hay un frente de Pareto y que cada frente, domina al anterior, así hasta llegar a uno de los extremos, donde el frente de Pareto no domina a otro vector o frente y cuando lleguemos al frente óptimo, el cual será la mejor solución, veamos un ejemplo.

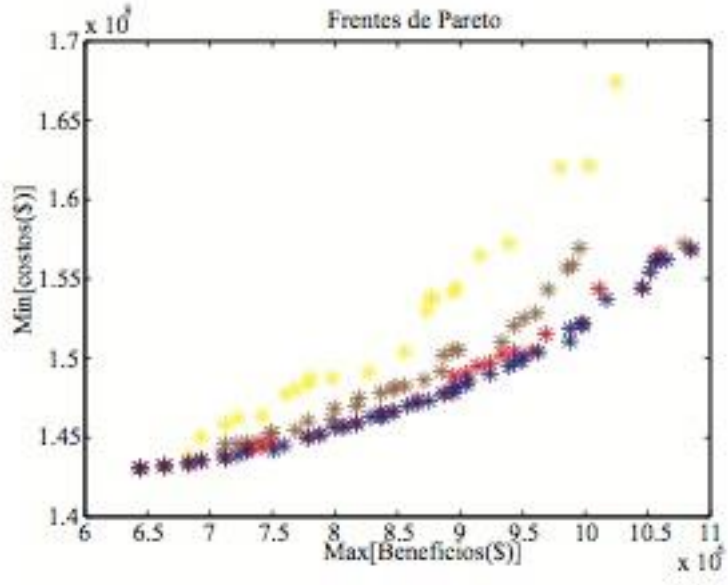


Figura 16: Representación de diferentes frentes de Pareto. Imagen sacada de http://www.scielo.org.co/scielo.php?pid=S1794-91652014000100005&script=sci_arttext.

En el frente de Pareto sabemos que debemos aproximarnos lo máximo posible a la zona óptima y que el conjunto de soluciones sea lo más uniforme posible, si no se cumple esto se deberá modificar algunas de las características del proyecto para que esto sea posible, ya que esa solución no nos será de utilidad.

Nótese como dependiendo del espacio considerado (espacio de decisión o espacio objetivo), el número de soluciones Pareto puede diferir. En el espacio objetivo, dos soluciones con los mismos vectores objetivos son consideradas el mismo punto, sin embargo, estas pueden representar dos

soluciones distintas en el espacio de decisión. El conjunto Pareto óptimo es denominado también conjunto mínimo Pareto óptimo completo mientras que representa el conjunto máximo Pareto óptimo en el espacio de decisión.

Es importante destacar que, idealmente, a uno le gustaría obtener una solución minimizando todos los objetivos. Supongamos que el óptimo para cada objetivo es conocido siendo optimizados por separado.

Definición Vector ideal. Un punto $y' = (y_1', y_2', \ldots, y_n')$ es un vector ideal si minimiza cada función objetivo x' en $F(x)$, es decir:

Ecuación 5: Fórmula del vector ideal si minimiza la función objetivo.

$$y_i' = min\big(f_i(x)\big), x \in S, i \in [1, n]$$

El vector ideal es generalmente una solución utópica en el sentido de que no suele ser una solución factible en el espacio de decisión. Sin embargo, algunos tomadores de decisiones definen un vector de referencia indicando el valor deseado en cada función objetivo. Esto generaliza el concepto de vector ideal. De este modo, el tomador de decisiones puede especificar ciertos niveles de aspiración a alcanzar en cada función f_i. Los niveles de aspiración indican los niveles de aceptación en el espacio objetivo. Una solución Pareto óptima debe satisfacer todos los niveles de aspiración y es denominada como una solución satisfactoria.

Definición Punto extremo. Un punto $\vec{v} = (v1, \ldots, vk)$ es un punto extremo si maximiza alguna función objetivo f_i en $F(x)$ en el conjunto Pareto, es decir:

Ecuación 6: Fórmula del punto extremo si maximiza toda la función objetivo del frente de pareto.

$$y_i' = max\big(fi(x)\big), x \in P', i \in [1, n]$$

Los puntos extremos e ideales proporcionan mucha información sobre el rango de valores del frente de Pareto óptimo figura 17.

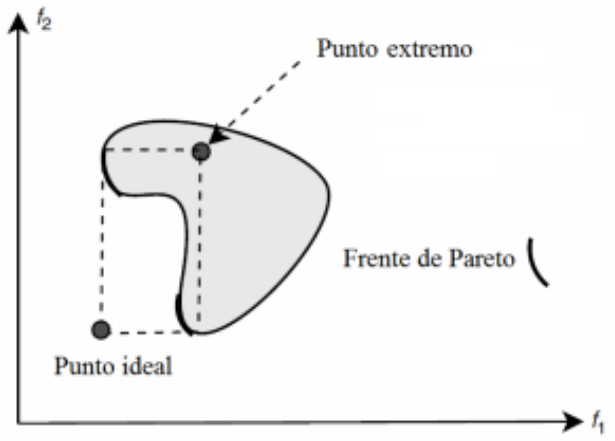


Figura 17: Representación del punto ideal y del punto extremo en problema de MO. Imagen sacada de David L. González – Álvarez – 2013 – Metaheurísticas, Optimización Multiobjetivo y Paralelismo para Descubrir Motifs en Secuencias de ADN.

Definición Función de utilidad. Una función (o valor) de utilidad v, la cual representa las preferencias del tomador de decisiones, convierte el vector objetivo en una función escalar:

Ecuación 7: Definición de utilidad de un valor o conjunto.

$$v: R^n \rightarrow R$$

La función de utilidad ha de ser minimizada para adaptarse a las preferencias del tomador de decisiones.

En optimización multiobjetivo, el concepto de mínimo local puede ser generalizado a una solución Pareto óptima local.

Definición Solución Pareto óptima local. Una solución X es Pareto óptima local si y solo si:

Ecuación 8: Definición de óptimo local de una solución.

$$\forall w \in N(x), F(w) nodominaa F(x)$$

4.6.3 Requisitos y clientes

Como hemos dicho reiteradas veces, el problema del siguiente lanzamiento consiste en seleccionar los requisitos que van a ser introducidos en ese lanzamiento, de esto se encarga el frente de Pareto, el de mostrarnos la solución más óptima siguiendo unos ciertos criterios. Los criterios son todos iguales de importantes, ya que se deben conseguir la máxima satisfacción con el menor costo posible, esto ya ha sido explicado anteriormente. El NRP es una forma de trabajar ágil y se usa para mejorar la producción del proyecto, así como el de puntos de control con los clientes.

Los diversos clientes que participan en el NRP tienen diferentes relevancias en el proyecto, esto nos indica que van a tener diferentes pesos (importancia), esto nos va a decir que cada uno va a influir en todos los requisitos de la siguiente entrega y hará que la prioridad de cada requisito varíe, claro está que saldrán más beneficiados los clientes que tengan mayor transcendencia en el proyecto. Estos requisitos tendrán un costo de desarrollo y cómo no, los recursos de la compañía son limitados, por tanto, los requisitos con mayores costos de producción no tendrán la máxima prioridad necesariamente, de este párrafo hemos sacado en conclusión que los requisitos serán seleccionados según su prioridad con los clientes y teniendo consideración el esfuerzo que conlleva realizar esa tarea.

Los requisitos también disponen de interrelaciones problemáticas, lo que es otra tarea más a la hora de resolver el frente de Pareto. Lo que nos indica que habrá requisitos que serán necesarios incluir si ya se han incluido otros o que sea necesario quitar requisitos si hay otros en la siguiente versión del proyecto. Las relaciones entre los requisitos, son restricciones para el proyecto. Todas las restricciones no son iguales como he ejemplificado antes, hay diferentes tipos de restricciones y las vamos a ver a continuación:

1. Implicación o precedencia (→): $[ri \Rightarrow rj]$ esto indica que un requisito ***ri*** no puede ser seleccionado si el requisito ***rj*** no ha sido seleccionado previamente.

2. Combinación o acoplamiento ($\oplus$): $[ri \oplus rj]$ significa que un requisito ***ri*** debe ser incluido obligatoriamente si el requisito ***rj*** está seleccionado.

3. Exclusión ($\otimes$): $[ri \otimes rj]$ nos indica que un requisito ***ri*** no se puede incluir si el requisito ***rj*** está seleccionado.

4. Modificación: el desarrollo del requisito ***ri*** implica que algún otro requisito modificará su coste de implementación o de satisfacción que brindan los clientes.

La selección de requisitos se toma como un MOOP donde se deben cumplir estos dos objetivos, primero maximizar la satisfacción y segundo, minimizar los costos, teniendo en cuenta los problemas restrictivos que existen entre la relación de los requisitos. Por lo tanto, el seleccionador de requisitos podrá seleccionar un conjunto de soluciones no dominadas (por el frente óptimo de pareto) en vez de la solución óptima. El seleccionador de los requisitos a realizar, escogerá la solución según las circunstancias que más influyan en ese momento.

4.7 Definición MO del problema de requisitos software

Disponemos de unas nomenclaturas, para identificar cada elemento en el proyecto, en MONPR (Multi-Objetive Next Release Problem), casi todos los datos vienen dado en vectores.

- R: $y' = (y_1', y_2', \ldots, y_n')$*serán los requisitos que van a implementarse en la siguiente entrega, el tamaño del vector son la cantidad de recursos de los que disponemos.*
- C: $C = (c_1, c_2, \cdots, c_n)$ *es prioridad que cada cliente cree que va a tener cada recurso, normalmente son matrices ya que se tienen varios clientes y cada cliente le da una prioridad. Quedaría tal que así:*

Ecuación 9: Representación matricial de los recursos y sus prioridades.

$$V = \begin{pmatrix} v11 & v12 & \cdots & v1n \\ \vdots & & \ddots & \vdots \\ vn1 & vn2 & \cdots & vnn \end{pmatrix}$$

Donde cada columna representa a cada requisito y donde cada fila representa a cada requisito:

- W: $W = (w_1, w_2, \cdots, w_n)$*representa el peso de cada cliente, es decir, la prioridad que tiene ese cliente, cada Wi representa a un cliente.*

- E: $F(x)$*esfuerzo que conlleva hacer cada requisito, cada "ei" representa al esfuerzo de cada requisito.*

- S: $S = (s_1, s_2, \cdots, s_n)$*satisfacción que conlleva hacer ese requisito, calcular la satisfacción de cada requisito influyen diversos factores, como la relevancia de cada cliente y la importancia que le da cada cliente a un requisito específico, por tanto, para calcular cada la satisfacción de cada requisito, se usa la siguiente fórmula:*

Ecuación 10: Cálculo de la satisfacción general del MOP.

$$S_j = \sum_{i=1}^{m} W_i * V_{ij}$$

El objetivo de MNORP es encontrar un conjunto de soluciones resolviendo los dos objetivos principales, minimizando el coste de desarrollo y maximizando la satisfacción del cliente, ¿pero cómo sabemos que es una solución? La solución es un subconjunto de R, es decir X:

- X: $X = (x_{1,}x_{2,}\ldots,x_n)$*vector de las decisiones que se va a llevar a cabo en la siguiente entrega, por tanto, los valores serán binarios, es decir, variaran entre 0 y 1, esto nos indica que si es un 0 no se realizará la tarea y si es un 1, sí se realizará la tarea. El vector tendrá el tamaño de la cantidad de requisitos que se vayan a tener en cuenta en la siguiente entrega.*

Unificando *X* con los interrelaciones de los requisitos, debemos tener en cuenta qué requisitos están presentes, pongamos un ejemplo, en la implicación, si ***ri*** está en X, es decir, si vale 1, **rj** debe estar también en X, lo mismo pasa con la combinación y con la exclusión.

Después de saber los requisitos que se van a implementar en la siguiente entrega, debemos calcular la satisfacción y esfuerzo que genera ese vector de solución, por tanto debemos usar las siguientes fórmulas para calcular estos valores.

Nuestro objetivo es maximizar la satisfacción, es decir, que el conjunto de la suma de todas las satisfacciones salga lo mayor posible:

Ecuación 11: Maximización de la satisfacción.

$$Maximize\ S(X) = \sum_{j \in X} s_j$$

A parte nuestro objetivo es e minimizar el esfuerzo, por consiguiente, la suma de todos los esfuerzos correspondientes a cada requisito, debe salir lo más bajo posible:

Ecuación 12: Minimización del esfuerzo.

$$Minimize\ E(X) = \sum_{j \in X} e_j$$

Ya que nuestro esfuerzo debe ser mínimo, será irremediablemente más alto de lo que obtenemos, además nuestro cliente tiene un límite de recursos, por eso debemos ponerle un límite a ese esfuerzo ***LC***. De tal manera que nuestro esfuerzo debe ser menor a ese límite.

Ecuación 13: limitación del esfuerzo.

$$\sum_{j \in X} ej < L_C$$

4.7.1 Codificación de la solución

Ya que nuestro proyecto entra utiliza la computación evolutiva y esta, necesita un individuo/solución para ese problema. Es importante un buen diseño del individuo, ya que va a ir evolucionando y es necesario una gestión rápida de él. La codificación de la solución debe dar toda la información necesaria para representar correctamente el problema de la selección de requisitos. En este caso, la solución se expresará con el vector ***X*** ya explicado previamente, es decir, con el conjunto de decisiones tomada, dicho de otra forma, la selección de requisitos en la siguiente entrega más los dos objetivos a cumplir en el problema, en nuestro caso, son la satisfacción del cliente **(S)** y el esfuerzo que conlleva a hacer dichas tareas **(E)**.

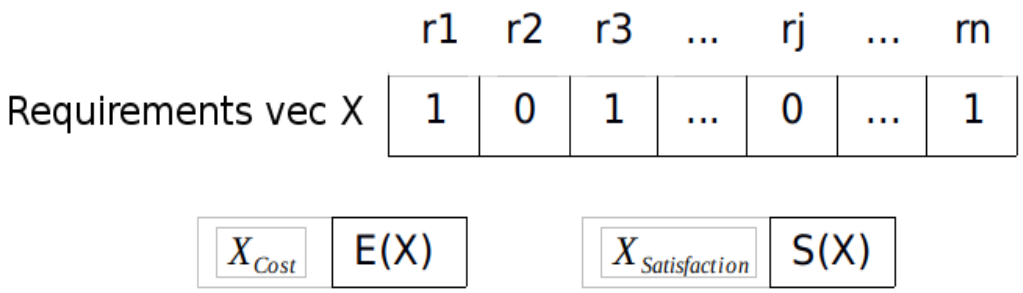


Figura 18: Conjunto de solución X, junto a su esfuerzo E y satisfacción S. Esta imagen ha sido realizada por el autor de este documento.

Si vemos las diferentes restricciones que hay entre estos requisitos, como la implicación, la combinación o la exclusión, nuestro algoritmo evolutivo(lo explicaremos más adelante) irá evolucionando y modificará reiteradas veces este vector **X**, de tal manera, que habrá que añadirle las restricciones al finalizar este algoritmo, por tanto, se corregirá con el conjunto de restricciones. Veremos un ejemplo con la combinación **(⊕)** y con la exclusión **(⊗)**.

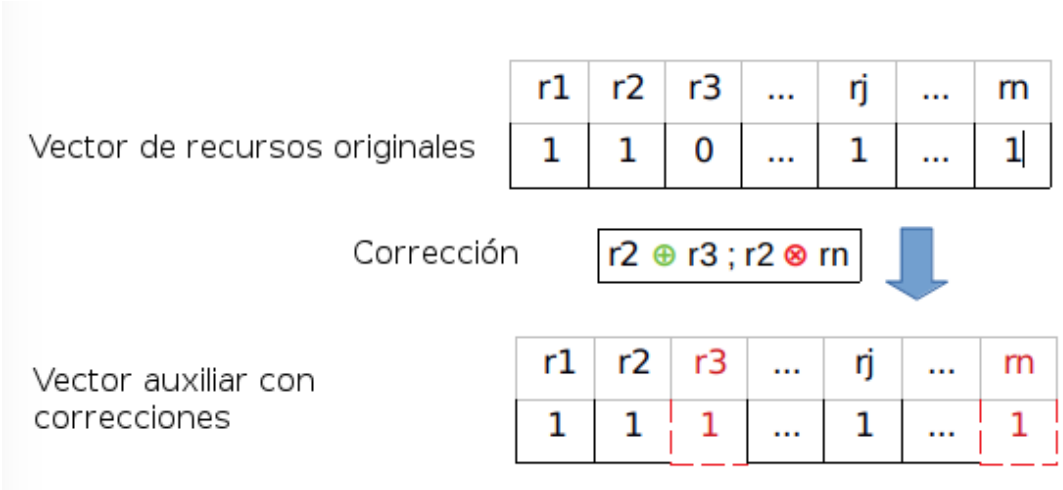


Figura 19: ajuste de las restricciones. Esta imagen ha sido realizada por el autor de este documento.

Una vez aplicada la corrección de las restricciones, debemos aplicar la restricción de esfuerzo/recursos utilizados, ya que es un límite que nos pone el cliente. En este caso tenemos un límite del **30%**, es decir, ***Lc<30%***.

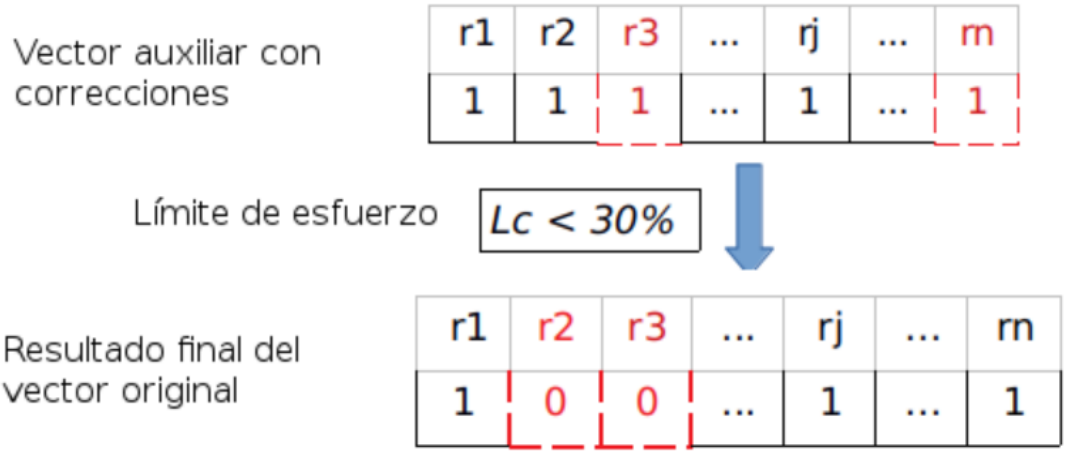


Figura 20: Ajuste de la restricción del esfuerzo. Esta imagen ha sido realizada por el autor de este documento.

Por tanto todas las soluciones serán de este estilo y sufrirán este tipo de modificaciones. También debemos de tener en cuenta de que el Límite de esfuerzo ***Lc*** puede variar entre los valores de 100%(sin límite de esfuerzo), 70%, 50% y 30%, pero eso es algo que veremos más adelante en el apartado de los resultados obtenidos (Apartado 6.2).

Si por algún motivo las restricciones del problema no se llevan a cabo correctamente, el resultado final puede variar notablemente, por tanto, ese resultado no sería válido ya que no se están aplicando correctamente las correcciones.

4.7.1 Métrica de la solución

Entre las numerosas métricas que existen para comprobar el rendimiento de los algoritmos utilizados para la optimización de problemas multiobjetivo, la más utilizada es la métrica del hipervolumen, o también conocida como Smetric o medida del Hipervolumen. La mayor parte de la información de este apartado ha sido obtenida de Pedro Manuel Ramos Rodríguez – 2017.

Ecuación 14: Función del cálculo del Hipervolumen.

$$HV(S,R) = volume \left(\bigcup_{i=i}^{|Q|} vi \right)$$

Lo que quiere decir esta fórmula, es que por cada solución $\vec{s_i} \in S$, un hipercubo $F(x)$ es construido con el conjunto de referencia y la solución $\vec{s_i}$ como la diagonal entre las esquinas de los hipercubos.

El hipervolumen es una métrica unitaria (recibe como parámetro un único conjunto A para ser evaluado) que mide cuánto del espacio objetivo es dominado por un conjunto no dominado. Para realizar el cálculo del hipervolumen y comprobar el espacio cubierto por el conjunto, se necesita un punto de referencia Figura 21. Este punto de referencia, normalmente se calcula con la peor solución del conjunto para los diferentes objetivos, es decir, si el frente dado esta normalizado (valores entre 0-1), el punto de referencia suele ser (1,1) en el caso de que sea un problema multiobjetivo con dos objetivos.

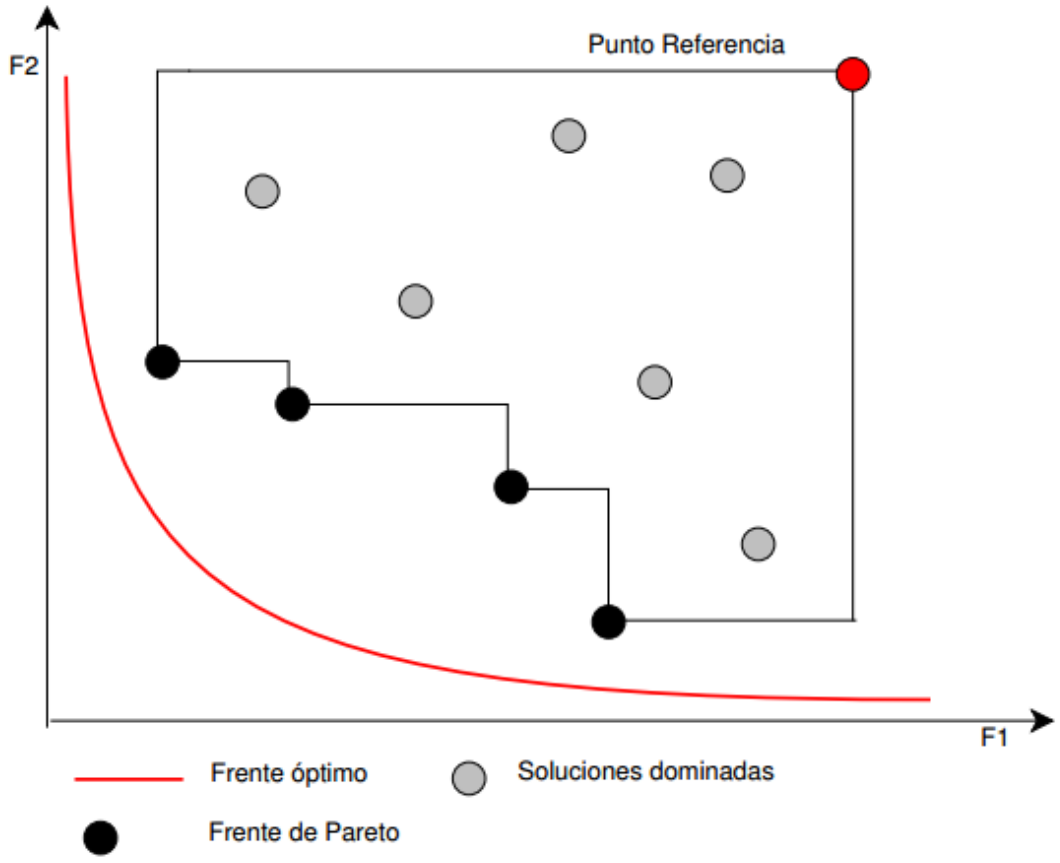


Figura 21: Punto de referencia para el cálculo del hipervolumen. Esta imagen ha sido sacada del Pedro Manuel Ramos Rodríguez – 2017 – Herramienta para la validación estadística de metaheurísticas.

Como se puede apreciar en la imagen, se posee una serie de soluciones dominadas, el punto de referencia, el frente de Pareto, el frente de Pareto óptimo y el hipervolumen resultante. El frente de Pareto es el conjunto de puntos o soluciones obtenidos, donde no existe otro punto que mejore el resultado de alguno de los objetivos sin empeorar el otro. Sin embargo, el frente de Pareto óptimo, sería el conjunto de soluciones más óptimas para resolver el problema dado. Llevando esto a la realidad es muy complicado conseguirlo y lo que se realiza es aproximar esta solución óptima, o lo que se denomina conjunto de aproximación.

5. MATERIAL Y MÉTODO

En este apartado seremos capaz de ver los recursos que han sido utilizados en este proyecto, tanto los recursos hardware, como los recursos software. Además podremos ver los métodos que se han seguido para que el proyecto se haya llevado a cabo.

5.1 Hardware y Software usados

En este apartado hablaremos sobre los recursos hardware necesario que necesitamos para poder ejecutar el algoritmo, además, hablaremos sobre cómo funcionan los recursos software de los que disponemos, así como del propio pseudocódigo.

5.1.1 Hardware empleado

El hardware empleado para ejecutar el algoritmo, ha sido el mismo, no ha variado a lo largo de todas las pruebas hechas. El hardware empleado ha sido mi equipo personal:

- El equipo dispone de un procesador intel core i7-2630QM, esto nos indica que es un procesador multicore, es decir, un procesador de 8 núcleos a 2.00 GHz cada núcleo.

- El equipo dispone de una capacidad de 31GB.El equipo dispone de una memoria principal de 8GB DDR3.

- Por último el equipo tiene un sistema operativo Linux, concretamente la versión de Linux 17.10.

5.1.1 Software empleado

El Software empleado ha sido enteramente realizado por el autor de este documento, exceptuando un código externo que me ha ayudado a calcular el Hipervolumen llamado “hyo_ind.c” perteneciente a implementa el indicador de hipervolumen unario como se propone en *“Zitzler, E., Thiele, L., Laumanns, M., Fonseca, C., and Grunert da Fonseca, V (2003): Performance Assessment of Multiobjective Optimizers: An Analysis and Review. IEEE Transactions on Evolutionary Computation, 7(2), 117-132”*.

El código **“hyo_ind.c”** funciona pasándole los parámetros de esfuerzo y satisfacción que obtenemos de los resultados, a partir de estos resultados, obtenemos un hipervolumen hemos-explicado en el apartado(4.7.1), a partir de este hipervolumen podemos obtener el frente de pareto y esa será nuestra representación del resultado. Para poder pasarle el esfuerzo y satisfacción que obtenemos de todo el cálculo de la población, debemos indicarle previamente en qué orden van los parámetros, es decir, nos calculará incorrectamente el hipervolumen, ya que ese hipervolumen será representado en un eje de gráficas.

La configuración será en un fichero a parte llamado: “hyp_ind_param.txt” la configuración viene dada en la Figura 22. Este fichero nos indica qué parámetro viene en cada eje de coordenadas, si el esfuerzo o la satisfacción.

```
dim <integer>
obj <+|-> <+|-> ...
method <0|1>
nadir <real> <real> ...
```

Figura 22: Archivo de configuración para calcular el Hipervolumen. Esta imagen ha sido realizada por el autor de este documento.

El primer parámetro nos indica qué dimensión ha de tener, el segundo es el valor dispuesto a minimizar si es + -, el primer valor es el que estamos dispuesto a maximizar y el segundo a minimizar, es decir, si tenemos primero a la satisfacción, estaremos maximizando la satisfacción y minimizando el esfuerzo, tendremos tantos + | – como dimensiones hayas a tener en cuenta,

el tercero es un método interno y por último, debemos indicar entre los dos valores, debemos indicar dónde está el punto máximo de hipervolumen, ese sería el punto ideal a llegar.

5.2 Multi-objetive Shuffled Frog Leaping Algorithm (MO-SFLA)

Ya hemos hablado del algoritmo de las ranas saltarinas antes en el apartado 4.5 por ahora nos centraremos en explicar el funcionamiento de dicho algoritmo, para ello, nos apoyaremos del pseudocódigo así todo quedará mucho más claro a la hora de explicar punto por punto qué va haciendo el algoritmo.

Como hemos dicho en el apartado algoritmos basados en población (Apartado 4.3), para todos estos algoritmos, es necesario generar nuestra población previamente antes de proceder a introducirnos más de lleno a explicar cómo funciona el algoritmo, todo lo que vamos a explicar del algoritmo, viene en la Figura 23, esta imagen es un pseudocódigo del algoritmo de las ranas, el pseudocódigo viene más desglosado en la Figura 24, utilizaremos estas dos imágenes de guías.

La población que hemos generado es una población inicial de 40, por tanto, deberemos de tener un número de charcos/memeplex para tener una población semejante en cada charco, por tanto si tenemos 4 charcos, tendremos una población de 10 ranas en cada charco y si tuviéramos una población de 100, en cada charco habría 25 ranas. Siempre deberemos de tener una población repartida equitativamente entre los charcos, ya que si no, nuestro algoritmo nos daría un error de ejecución, es decir, que el número de charcos ha de ser un número divisor de la población.

```
Begin;
  Generate random population of P solutions (individuals);
  For each individual i ϵ P: calculate fitness (i);
  Sort the whole population P in descending order of their
    fitness;
    Divide the population P into m memeplexes;
    For each memeplex;
        Determine the best and worst individuals;
        Improve the worst individual position using Eqs. 1
        and 2;
        Repeat for a specific number of iterations;
    End;
    Combine the evolved memeplexes;
    Sort the population P in descending order of their
    fitness;
  Check if termination = true;
End;
```

Figura 23: Pseudocódigo del algoritmo de las ranas (MOSFLA). Esta imagen ha sido sustraída Del EMAD ELBELTAGI, TAREK HEGAZY and DONALD GRIERSON – 2005 – A modified shuffled frog-leaping optimization algorithm: applications to project management.

La población siempre es generada de manera aleatoria, por eso, en cada ejecución, nunca se tendrá la misma población. Cuando se genere la población, también se debería de haber decidido el número de charcos y el número de interacciones que va a haber en los charcos.

Una vez generada la población se debe calcular el fitness de cada individuo de la población o rana, esto quiere indicar que se calculará la satisfacción y el esfuerzo de cada individuo, normalmente cada individuo nos dará resultados diferentes, habrá mejores individuos y también los habrá peores. Calculamos el fitness para poder ordenarlos correctamente de mejor a menor individuo, para saber si un individuo es mejor que otro, debemos saber si este es dominado por otros individuos, además de saber qué individuos dominan a quien, debemos de tener en cuenta otro factor de ordenación que se explicará más detenidamente en el apartado 5.2.1.

Una vez ordenados los individuos, lo que debemos hacer es repartirlos por los diferentes charcos, pero no podemos poner los 8 o 10 primeros en el

primer charco y los 8 o 10 últimos, en el último charco, entonces solo se mejoraría una parte de la población. El método de ordenación a seguir es un método por barajadura, es decir, si disponemos de 5 charcos y 40 ranas, repartiremos las 5 primeras ranas entre los cinco charcos, es decir, la primera rana/individuo al primer charco, la segunda rana al segundo charco, así hasta llegar al último charco, una vez que se han repartido los cinco primeros, se reparten los cinco siguientes entre los mismos cinco charcos, de la misma manera hasta acabar con los individuos. Este método de distribución de individuos hace que los individuos malos puedan mezclarse con los buenos individuos, así se consigue un mayor alto porcentaje de aprendizaje (esto es algo que veremos en los párrafos siguientes).

La siguiente parte que vamos a explicar viene dada en la figura 24 y es el esquema de la derecha, esta parte se encarga de explicar la “búsqueda local”, es decir, la mejora de los individuos.

Para hacer la mejora de los individuos, se debe de seguir los siguientes criterios. Antes de nada, todo esto se hace desde los propios charcos.

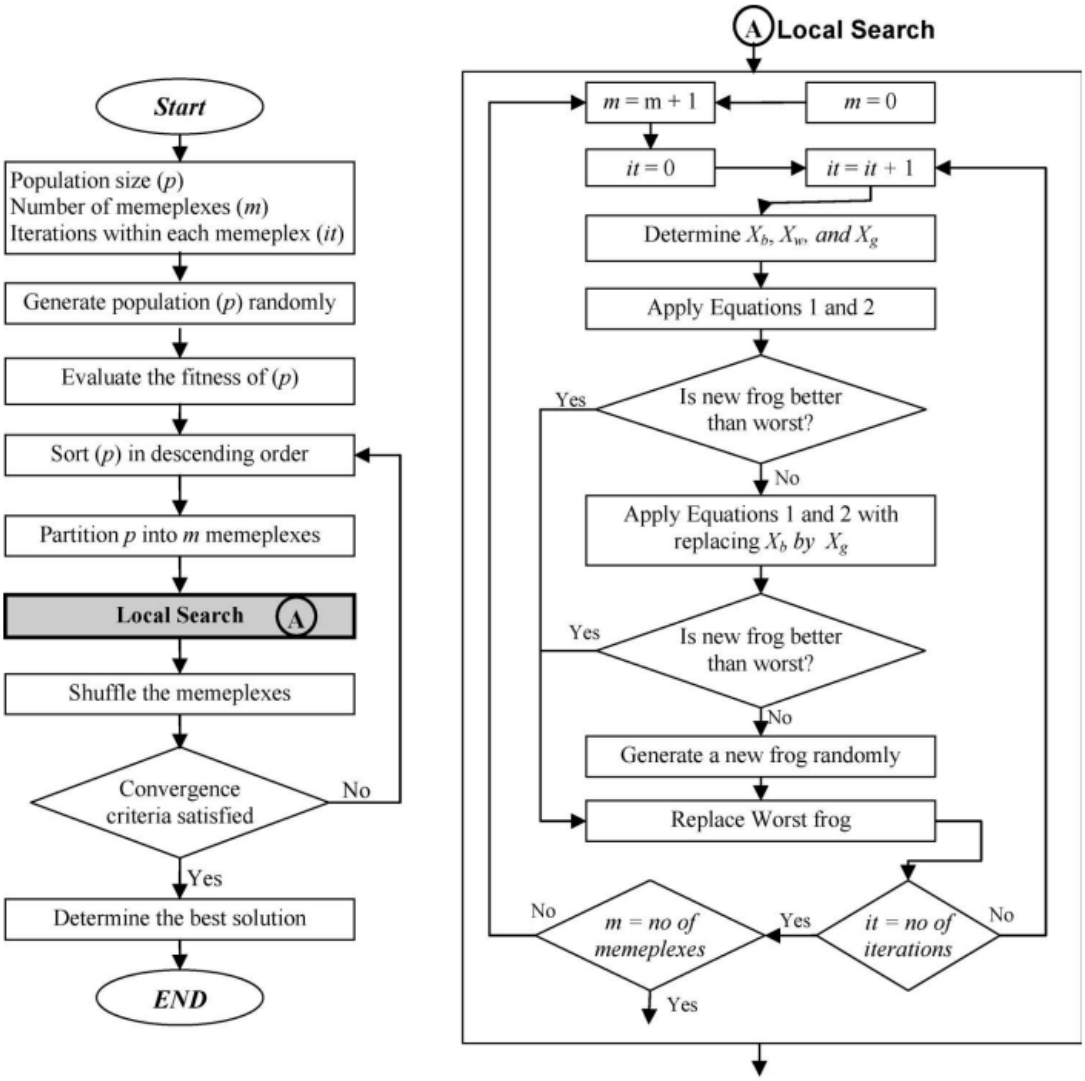


Figura 24: esquematización del pseudocódigo incluyendo el bucle mejorar. Esta imagen ha sido sacada de EMAD ELBELTAGI, TAREK HEGAZY and DONALD GRIERSON – 2005 – A modified shuffled frog-leaping optimization algorithm: applications to project management.

En primer lugar, se deben de localizar el mejor individuo del charco, no habrá problemas de identificarlo, ya que es el primer individuo del charco, lo siguiente será tener localizado, que siguiendo el criterio anterior, también será fácil de encontrar, ya que será el último individuo añadido al charco. Ya por último se debe de tener localizado el mejor global, este individuo será también fácil de encontrar, ya que se encuentra en la primera posición del primer charco.

En cada pasado por el charco, el peor individuo en ese momento pasará por las siguientes etapas. Lo primero que se hará es intentar que la peor rana intente aprender de la mejor, el criterio que debe sigue es el siguiente:

Ecuación 15: criterio de cambio propuesto por el pseudocódigo.

$$Changefrogposition(D_i) = rand(X) * (Xb - Xw)$$

Este es el criterio propuesto, no es el que nos resulta de utilidad, nosotros al trabajar con valores binarios, podemos obtener valores negativos y esos valores, pueden variar mucho los resultados de la práctica de manera negativa, por tanto, nuestro criterio es el siguiente:

El criterio anterior solo es válido si tenemos números enteros y no son binarios, no es nuestro caso. Debemos adaptarnos a nuestra situación, en la que estamos limitados por los valores binarios. La adaptación realizada consiste en modificar el peor individuo e intentar mejorarlo en los dos sentidos, en caso de que no pueda ser mejorado o mejor dicho que el nuevo individuo no domine al peor hallado, ya que nuestro modo de optimización es paralelo, esto fue explicado en el apartado de algoritmos basados en población (Apartado 4.3).

Lo que intentamos es una de las tres opciones de las que disponemos: eliminar un requisito, sustituir un requisito por otro o añadir un requisito. Estas tres opciones se van a tener en cuenta a la hora de intentar mejorar los recursos de in individuo.

El anterior proceso lo repetimos dos veces ya que es posible que se pueda mejorar en cualquiera de las tres posibilidades dadas. Pero en última instancia en caso de que el individuo no haya podido ser mejorado. Lo que sucederá es que generaremos un nuevo individuo de manera aleatoria, obviamente le calcularemos su fitness.

Una vez mejorado o generado el nuevo individuo, lo que se hará es una ordenación parcial de este en el mismo charco, por tanto lo que se hará es saber a cuál de los individuos domina para poder colocarlo correctamente.

Una vez pasado por todos estos pasos, se vuelven a recoger los individuos de los charcos para así obtener una población mejorada a partir de la inicial generada.

5.2.1 Non-dominated Sorting Genetic Algorithm-II

Esta parte se encarga de ordenar la población de ranas de las que disponemos, la Figura 25 nos muestra el pseudocódigo de sobre cómo funciona el método de ordenación.

```
fast-non-dominated-sort(P)
for each p ∈ P
   S_p = ∅
   n_p = 0
   for each q ∈ P
      if (p ≺ q) then                    If p dominates q
         S_p = S_p ∪ {q}                 Add q to the set of solutions dominated by p
      else if (q ≺ p) then
         n_p = n_p + 1                   Increment the domination counter of p
   if n_p = 0 then                       p belongs to the first front
      p_rank = 1
      F_1 = F_1 ∪ {p}
i = 1                                    Initialize the front counter
while F_i ≠ ∅
   Q = ∅                                 Used to store the members of the next front
   for each p ∈ F_i
      for each q ∈ S_p
         n_q = n_q − 1
         if n_q = 0 then                 q belongs to the next front
            q_rank = i + 1
            Q = Q ∪ {q}
   i = i + 1
   F_i = Q
```

Figura 25: Pseudocódigo sobre el cálculo de la distancia crowding. Imagen sacada de EMAD ELBELTAGI, TAREK HEGAZY and DONALD GRIERSON – 2005 – A modified shuffled frog-leaping optimization algorithm: applications to project management.

Este algoritmo se va a encargar de organizar la población por frentes o dicho de otra manera, por rangos. Irá elemento a elemento de la población e irá comprobando si este puede dominar al resto de la población.
Por tanto podemos tener varios casos:

- El primer caso es que ese individuo domine a parte de la población.

- El segundo caso es que el individuo no sea capaz de dominar a cierta parte de la población.

- Por último y tercer caso es que ese individuo sea dominado por cierta parte de la población.

Si pueden dar varios casos a la vez, incluso los tres al mismo tiempo. Pongamos un ejemplo para entenderlo mejor: dado un individuo p, este se irá comparando con toda la población, este dominará a gran parte de la población, además no será dominado por ningún individuo, pero habrá un sector en concreto que no será capaz de dominar, este sector, se considerará que es el primer frente de pareto, a partir de ahí no tendremos en cuenta ese sector de la población para formar el siguiente frente o rango. De tal manera que nos quedará una población organizada por grados como se ve en la Figura 26. En nuestro código, sería un vector de población y ese vector iría ordenado de mejor frente a peor frente y a su vez, cada frente o sección de vector, iría ordenado por dominancia.

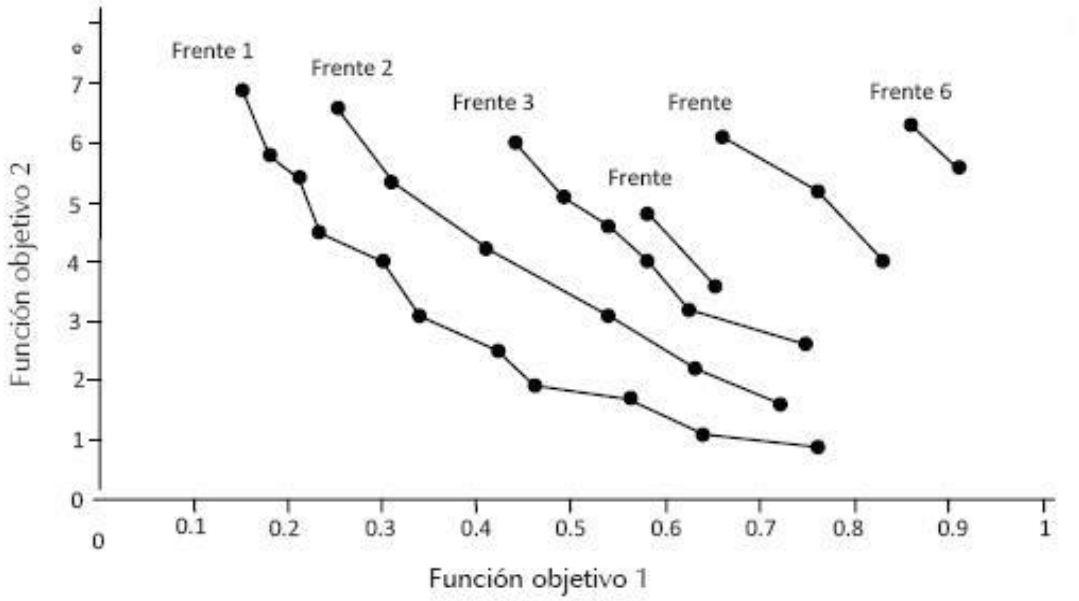


Figura 26: Representación de diferentes frentes de pareto. Esta imagen ha sido obtenida de http://www.scielo.org.mx/scielo.php?pid=S1405-77432014000100012&script=sci_arttext.

Una vez calculado los frentes y divididos por rankings, debemos apoyarnos en el cálculo de la distancia crowding. La medida de crowding se utiliza para seleccionar las soluciones más dispersas entre los individuos del último frente utilizado en la nueva población. Cuanto mayor sea la distancia de crowding de una solución al resto de su frente mejor, ya que hay menos concentración en esa zona.

En la Figura 27 se puede apreciar gráficamente cómo se calcula este atributo de forma gráfica.

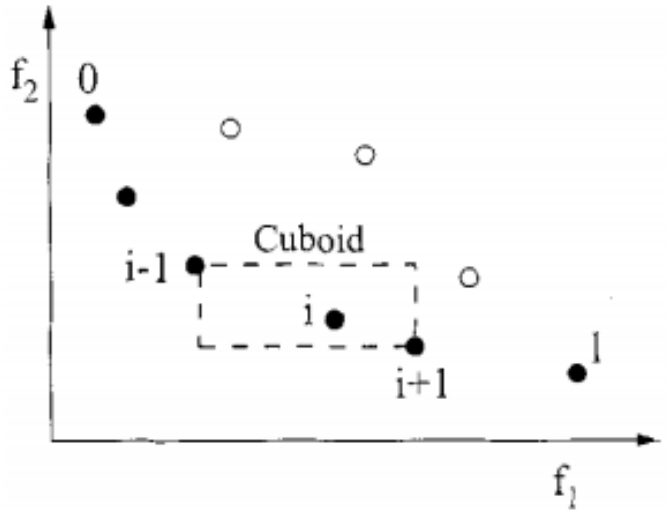


Figura 27: Representación gráfica del hipervolumen. Esta imagen ha sido sacada de Kalyanmoy Deb, *Associate Member, IEEE*, Amrit Pratap, Sameer Agarwal, and T. Meyarivan – 2002 – A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II

Para calcular la distancia crowding se tienen en cuenta varios factores, si se dispone de más de un objetivo, se debe hacer este proceso tantas veces como objetivo a satisfacer, este caso, solo disponemos de esfuerzo y satisfacción, por lo tanto, solo debemos hacerlo dos veces.

Veremos cómo se hace para el esfuerzo, ya que para la satisfacción es idénticamente igual: Una vez hallados los rankings se ordenan como tal, se va por cada rango teniendo en cuenta el esfuerzo máximo y el rango mínimo que se puede alcanzar, empezamos por el primer elemento y calcularemos la diferencia con respecto al esfuerzo con el anterior y el siguiente, si son puntos extremos, se calcularan con los máximos posibles.

En la Figura 28 podemos ver es pseudocódigo encargado de calcular la distancia crowding.

```
crowding-distance-assignment(I)
l = |I|                                   number of solutions in I
for each i, set I[i]_distance = 0         initialize distance
for each objective m
  I = sort(I, m)                          sort using each objective value
  I[1]_distance = I[l]_distance = ∞       so that boundary points are always selected
  for i = 2 to (l − 1)                    for all other points
    I[i]_distance = I[i]_distance + (I[i+1].m − I[i−1].m)/(f_m^max − f_m^min)
```

Figura 28: Pesudocódigo encargado de calcular la distancia crowding. Esta imagen se ha sacado de Kalyanmoy Deb, *Associate Member*, *IEEE*, Amrit Pratap, Sameer Agarwal, and T. Meyarivan – 2002 – A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II.

5.2.2 Filtración de un resultado

La solución que guardamos es bastante grande, ya que guardamos resultados cada vez que generamos un individuo aleatorio, por tanto la cantidad de resultados a manejar de la que se dispone es bastante grande, por tanto, para facilitarnos las cosas, lo que debemos hacer es filtrarnos ese resultado. De tal manera que lo que ocurra es quedarnos con una cantidad de resultados que dominen al resto de la población bastante reducida. Si disponemos de más de unos 10000 resultados a analizar, la cantidad de frentes serían enormes, por tanto nuestro objetivo es quedarnos con el frente dominante.

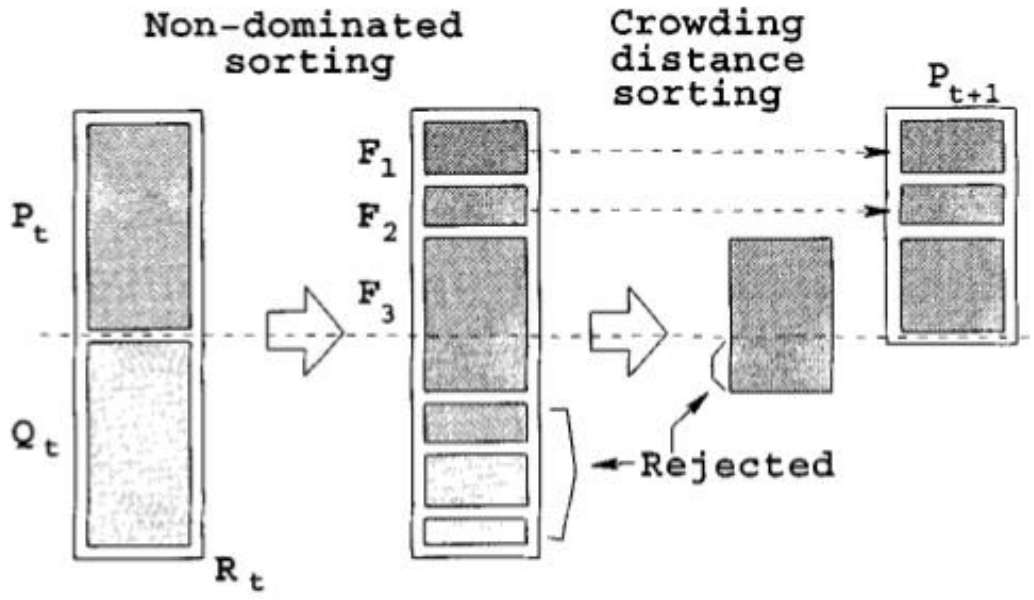


Figura 29: Selección de la población que no es dominada. Esta imagen ha sido sacada de Kalyanmoy Deb, *Associate Member*, *IEEE*, Amrit Pratap, Sameer Agarwal, and T. Meyarivan – 2002 – A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II.

En parte lo que hace NGSA-II es recortar la población para quedarnos con los individuos dominantes de la población, por tanto nos quedaremos con unos 30 resultados por cada prueba, por tanto, analizar esta cantidad de resultados es mucho más liviano que analizar 10000.

6. RESULTADOS Y DISCUSIÓN

En este apartado veremos los diferentes ajustes que hemos realizado para mejorar los resultados que hemos obtenido al principio y así tener unos valores competentes.

6.1 Ajustes del algoritmo

En esta sección de la documentación vamos a analizar los resultados obtenidos por nuestro algoritmo, además de las distintas modificaciones que ha sufrido y los resultados de estas mismas.

Como muy bien se ha repetido a lo largo de la documentación, las pruebas se van a realizar con el algoritmo MO-SFLA junto al NSGA-II, la cantidad de veces que hemos ejecutado el algoritmo ha sido un total de 31 veces, ya que a la hora de hacer la media y sobre todo la mediana podamos elegir un valor que está justamente en la mitad de todo los valores obtenidos. Los resultados los vamos a medir mediante el Hipervolumen que obtengamos de cada ejecución, por tanto, los resultados finales de cada ejecución que veremos será la media del Hipervolumen, la mediana, la media de resultados obtenidos y por último el frete de pareto obtenido por esta población.

La cantidad de evaluaciones que se van a hacer dentro del MO-SFLA son un total de 10000, ya que es la cantidad de pruebas que tomamos como referencia para poder comparar en un futuro nuestros resultados. Estas 10000 evaluaciones serán repartidas por la cantidad de evaluaciones que se realizarán por charco, el número de charcos y las propias repeticiones del algoritmo de las ranas.

La configuración usada en casi todas las pruebas que veremos a continuación va a ser siempre la misma Figura 30: si en algún momento cambia, se explicará dónde cambia.

La Figura 30 junto a la Figura 31 es la configuración por defecto que vamos a tener para todas la pruebas y modificaciones llevadas a cabo en el algoritmo, está compuesta por la prioridad que le pone cada cliente (cl), el esfuerzo que con lleva realizar la tarea (Effort), las restricciones (Interactions) y por último la prioridad de cada cliente (clients weigth).

		r_1	r_2	r_3	r_4	r_5	r_6	r_7	r_8	r_9	r_{10}	r_{11}	r_{12}	r_{13}	r_{14}	r_{15}	r_{16}	r_{17}	r_{18}	r_{19}	r_{20}
Effort		1	4	2	3	4	7	10	2	1	3	2	5	8	2	1	4	10	4	8	4
Priority level	cl_1	4	2	1	2	5	5	2	4	4	4	2	3	4	2	4	4	4	1	3	2
	cl_2	4	4	2	2	4	5	1	4	4	5	2	3	2	4	4	2	3	2	3	1
	cl_3	5	3	3	3	4	5	2	4	4	4	2	4	1	5	4	1	2	3	3	2
	cl_4	4	5	2	3	3	4	2	4	2	3	5	2	3	2	4	3	5	4	3	2
	cl_5	5	4	2	4	5	4	2	4	5	2	4	5	3	4	4	1	1	2	4	1
Interactions	$r_4 \Rightarrow r_8$ $r_4 \Rightarrow r_{17}$ $r_8 \Rightarrow r_{17}$ $r_9 \Rightarrow r_3$ $r_9 \Rightarrow r_6$ $r_9 \Rightarrow r_{12}$ $r_9 \Rightarrow r_{19}$ $r_{11} \Rightarrow r_{19}$ $r_3 \oplus r_{12}$ $r_{11} \oplus r_{13}$																				

Figura 30: Tabla de requisitos con los requisitos del proyecto, sus respectivos esfuerzos y la satisfacción que causa a cada cliente y sus respectivas restricciones. Esta imagen ha sido sacada de José M. Chaves-González , Miguel A. Pérez-Toledano – 2015 – Differential evolution with Pareto tournament for the multi-objective next release problem.

Clients' weights	cl_1	cl_2	cl_3	cl_4	cl_5
Dataset 1	1	4	2	3	4

Figura 31: Peso de los clientes. Imagen sacada de José M. Chaves-González , Miguel A. Pérez-Toledano – 2015 – Differential evolution with Pareto tournament for the multi-objective next release problem.

6.1.1 Tasa de aprendizaje del mejor individuo

Inicialmente en el algoritmo de mejora implementaba que el peor aprendiese del mejor y en caso de que no pudiera mejorarse, que intentara aprender del mejor global y si no, se creará un nuevo individuo de manera aleatoria. La forma que tenía de aprender del mejor o mejor global era generar un número aleatorio entre 0 y 100, es decir, nos movemos en un segmento porcentual del 0% al 100% y si salía un número de aprendizaje menor a una cierta cantidad entre estos dos parámetros, se cambiaría, la fórmula quedaría de la siguiente manera:

Esta fórmula la íbamos implementando en cada recurso del individuo, en caso de que X superase el %Aprendizaje, no cambiará peor(i) y se quedará

como está, en caso de que X si sea menor qué %Aprendizaje, peor(i) será sustituido por el mejor correspondiente.

Hemos probado este tanto por ciento con distintos rangos: con una tasa de aprendizaje del 25%, del 50%, del 75% y del 100%(siempre aprende del mejor).

Tabla 1: resultados del porcentaje de aprendizaje.

Tasa de aprendizaje	*Media HV*	*Media de resultados*
25%	59.8154290322581	27.741935483871
50%	59.9425129032258	26.5806451612903
75%	59.4584903225806	27.0967741935484
100%	59.1307612903226	26.9677419354839

Como podemos ver en la tabla, porcentaje de aprendizaje que se opta en los diversos casos, no influye mucho en el resultado, más bien depende del cómo se haya formado la población aleatoriamente, por tanto, esta modificación la quitaremos en un futuro ya que no nos influye en nada sobre el los resultados.

6.1.2 Mejorar Individuo

Para que nuestro algoritmo mejore, en primera instancia lo que debería ocurrir, era mejorar el individuo. En el apartado anterior lo que se intentaba era que el individuo mejorara a través del mejor individuo que haya en ese meme/charco o intentarlo con el mejor global. Esto se hacía en el apartado anterior, pero ha cambiado, de tal manera que ahora lo que se intenta mejorar es el individuo de manera que o solo se maximice o que el esfuerzo se minimice.

En este apartado utilizamos un random para decidir qué debemos seleccionar el random es fácil e intuitivo, ya que solo elige entre dos valores $r = rand(X)\%2$, por tanto, la población se mejora equitativamente entre el

esfuerzo y la satisfacción, así no hay desbalance de mejora ya que nuestro problema es un problema de multiobjetivo y no de mono objetivo, nuestro objetivo es siempre conseguir la máxima satisfacción, pero no olvidemos que se ha de conseguir con el menor esfuerzo posible, si no tuviéramos restricciones, se satisfaría a todos los clientes.

Tabla 2: resultados de la modificación de la mejora del individuo.

0	59,1634	26
1	60,3175	26
2	60,8695	28
3	60,718	27
4	59,975	26
5	58,3914	27
6	59,1358	29
7	59,0514	27
8	60,2674	26
9	59,6338	26
10	59,7484	28
11	61,49	23
12	60,971	25
13	59,8446	23
14	60,8142	27
15	59,6483	24
16	60,6508	23
17	62,2686	26
18	61,0487	28
19	60,5112	26
20	60,3465	23
21	60,3333	25
22	59,4981	23
23	60,6073	31
24	61,0329	26
25	60,8194	31
26	60,4097	26
27	59,6219	24
28	60,9117	25
29	59,7905	26
30	60,1607	29
Media	**60,25970968**	**26,12903226**
Mediana	**60,3333**	

De lo que se encarga esta modificación es de generar un aleatorio y según salga ese aleatorio, este aleatorio, elegirá entre esfuerzo o satisfacción, si sale modificar la satisfacción, tocará modificar un recurso cuyo satisfacción sea

la mínima y se intentará buscar un recurso que este tenga la misma o mayor satisfacción, o se intentará buscar el recurso con mayor satisfacción y se buscará un recurso con un esfuerzo menor. En caso de que el random elija modificar el esfuerzo, este tendrá que tener en cuenta las siguientes dos opciones. La primera es que busca el recurso con el mayor esfuerzo activo, este buscará un recurso similar que tenga menos esfuerzo y la misma o mayor satisfacción. En caso de que no encuentre nada, intentará buscar el recurso con menor esfuerzo e intentará sustituirlo por uno de mayor satisfacción.

6.1.3 Mutación de un recurso

En este caso, nos centraremos en las ideas del apartado anterior, pero esta vez se ha refinado el método, esto hace que nos incremente un poco el Hipervolumen. Ahora la metodología usada para mejorar el individuo funciona de la siguiente manera: se sigue el mismo criterio que en el apartado anterior, disponemos de un random el cual se encarga de decidir si mejorar la satisfacción o el esfuerzo. No seguimos ningún teorema matemático para mejorar cualquiera de las dos opciones.

Situándose en el caso de que nos toque mejorar la satisfacción, esta seguirá los siguientes pasos:

- Buscamos el requisito del individuo con la menor satisfacción, en caso de que no se pueda mejorar con respecto al esfuerzo, pero eso es algo que se verá unos párrafos más adelante.

- El siguiente paso que se seguirá, será el de busca el requisito no usado, es decir, que no este activo con la máxima satisfacción sin empeorar el esfuerzo.

- Si encontramos un requisito el cual sea capaz de maximizar sin perjudicar al esfuerzo, el requisito será sustituido por este. En caso de que no se haya podido hallar el requisito con estas características, se desactivará ese requisito, así pasaremos al siguiente requisito con

mayor satisfacción después del primero. Desactivamos los requisitos porque a la hora de buscar un recurso que cumpla nuestras especificaciones no interfieran.

- Seguiremos este proceso hasta que hayamos acabado con todos los recursos, en caso de que no hayamos encontrado ningún requisito que cumpla estas características, se procederá a activar todos los recursos que se habían desactivado(en caso de que se haya encontrado, también se activan)

- Si no se han encontrado ningún requisito, pasaremos a intentar mejorar el esfuerzo. Para mejorar el esfuerzo sigue el mismo criterio que la satisfacción, pero esta vez varía, esta vez lo que intentamos es buscar el requisito con mayor esfuerzo y sustituirlo por uno que tenga el mínimo esfuerzo con la máxima satisfacción posible.

- En caso de que no hubiéramos empezado por la satisfacción, hubiéramos empezado por el esfuerzo y si no hubiésemos encontrado un requisito en el esfuerzo, pasaríamos a la satisfacción.

A parte de esta modificación hemos añadido otra modificación que afecta de manera positiva a los resultados de este, la modificación consiste en mutar recursos al azar del individuo, pero dependen de una probabilidad para mutar, ya que siempre no mutan, por tanto, habrá recursos que antes si hacían y ahora no, otros que no se llevaban a cabo, pero ahora sí y por último habrá requisitos que no cambien. Esta probabilidad de mutar es de un **10%,** ya que la probabilidad de mutar una parte que un individuo posee es bastante baja.

Ya que nuestras dos opciones de intentar mejorar los individuos son buenas, lo que haremos será implementar otro random para cambiar entre estas dos opciones. Los resultados son los siguientes:

Tabla 3: resultados de la modificación mutación de un recurso.

0	60,6245	26
1	59,2622	25
2	59,2477	25
3	60,5612	26
4	60,3188	28
5	59,7247	23
6	60,5981	28
7	58,0357	26
8	60,494	23
9	60,8814	28
10	61,3174	23
11	59,3742	23
12	58,8209	31
13	60,4927	26
14	61,4176	24
15	59,4493	28
16	59,2767	26
17	60,502	27
18	60,303	23
19	61,1514	27
20	60,9657	27
21	59,2675	26
22	59,448	29
23	60,1502	26
24	59,8393	25
25	60,5467	27
26	59,8432	29
27	60,9841	25
28	60,6679	24
29	59,9776	24
30	59,307	25
Media	**60,09195806**	**25,90322581**
Mediana	**60,303**	

Como podemos observar la media de Hipervolumen ha disminuido unas décimas, pero la media de la cantidad de resultados ha aumentado con respecto a la anterior prueba, lo cual es positivo, ahora el objetivo es aumentar el Hipervolumen.

Se ha modificado la probabilidad que tiene de mutar un recurso del individuo, se ha probado con distintos porcentajes:

Tabla 4: resultados de la tasa de mutación.

Tasa de mutación	*Media HV*	*Media de resultados*
5%	60.3727096774194	26.5806451612903
10%	60.2417258064516	27.1935483870968
20%	60.6016129032258	26.4516129032258
40%	60.5337806451613	26.8387096774194

Se puede ver que si la tasa de mutación es del **20%** obtenemos un hipervolumen más alto, pero menos resultados.

6.1.4 Modificación de las restricciones

En el siguiente ajuste que se le ha realizado al algoritmo, nos centraremos principalmente en el modo en el que se realizan las restricciones del algoritmo, es decir, se modifican en el modo que se aplican la implicación, la combinación y la exclusión.

El método de aplicar estos tres métodos diferentes, de interactuar con los recursos, fue explicado en el apartado 4.6.3. Cambiaremos la forma en que estos se afectan entre ello sin perder la esencia de estos cambios.

En esta parte se percató que cualquier modificación, por mínima que fuese los resultados que daban, hacía que subiese el Hipervolumen de manera notoria, pero esas modificaciones estaban erróneas, ya que esas modificaciones no cumplían con las características acordadas ya explicadas en el apartado 4.6.3. La propuesta que se va a explicar a continuación es algo enrevesada, pero afecta a nuestros valores de manera positiva, tanto en el aumento del Hipervolumen como en el aumento de la media de valores.

La modificación consiste en que antes se hacía recorrer todas las restricciones y aplicarlas con normalidad, ahora se va a tener en cuenta y ver qué ocurre cuando no se cumplen esas restricciones, una de ellas es la

implicación lo que ocurre es que vamos a ver si es posible la interacción cuando hay una implicación.

Refresquemos un poco cómo funcionaba, la implicación hacía que si un recurso **A** estaba activo, este condicionaba a otro recurso **B** para que estuviera activo.

Ahora lo que se hace es buscar los recursos que estén condicionados por combinación y si el recurso **B** que se esté implicando en ese momento, se encuentra dentro de las restricciones de combinación, el recurso **A** será desactivado, este será desactivado ya que no será necesario para activar el recurso **B**. Los resultados que se han obtenido, son los siguientes:

En el caso de la combinación, si los requisitos no eran iguales y alguno de los dos estaba activo, lo que ahora sucede es que el requisito **A** que implicaba al otro **B**, ahora son desactivados.

Los resultados que se pueden observar en la tabla 5 son mejores con respecto a los anteriores obtenidos. Si nos fijamos más atentamente, el Hipervolumen ha subido notoriamente, además la cantidad de resultados no dominados ha aumentado con respecto a la anterior prueba.

Tabla 5: resultados de la modificación de las restricciones.

0	60,3478	29
1	61,8879	34
2	61,5862	31
3	61,1422	30
4	61,1607	32
5	61,436	31
6	61,8233	35
7	61,2595	30
8	62,1567	32
9	61,4874	36
10	61,44	35
11	61,1014	31
12	61,4637	31
13	61,797	26
14	61,8365	29
15	61,3504	32
16	61,3398	31
17	61,5295	33
18	61,436	27
19	60,8853	31
20	61,2055	29
21	60,8946	36
22	61,548	35
23	61,8787	34
24	61,5796	34
25	61,299	29
26	62,0302	37
27	61,7311	31
28	61,1225	36
29	61,407	30
30	61,8589	33
Media	**61,45233548**	**31,93548387**
Mediana	**61,44**	

6.1.5 Errores de mutación en la población

Una de las modificaciones que se han llevado a cabo, ha sido la de admitir errores en la población, esto conlleva a que se activen recursos de los individuos que antes no eran posibles, ya que antes cada vez que modificábamos un individuo, siempre debíamos aplicarle las restricciones acordadas.

Los errores estarán orientados sobre todo a la correcciones de en las restricciones propuestas, es decir, se pasarán por alto, esto quiere decir que

las implicaciones y las combinaciones no serían llevadas a cabo. Solamente se rectificará la población cuando ya se sepa que no se van a hacer más modificaciones en ella misma, por consiguiente, al final se quedará una población libre de errores.

Esta modificación se aplicará junto a las anteriores, ya que son beneficiosas para para nuestros resultados. A continuación veremos los resultados que se han obtenido de esta última modificación.

Tabla 6: resultados de la modificación de errores en la mutación.

0	61,5994	32
1	62,5282	37
2	62,5716	39
3	62,2001	35
4	62,5861	38
5	62,3635	34
6	62,32	39
7	62,3464	38
8	60,9328	32
9	62,2673	34
10	62,4794	36
11	62,4136	37
12	62,5624	35
13	61,8734	34
14	62,2252	41
15	62,8602	35
16	62,5018	33
17	62,3622	35
18	62,179	41
19	61,8642	34
20	61,9511	36
21	62,4636	38
22	62,822	35
23	62,7851	37
24	62,6625	33
25	62,656	37
26	62,7469	38
27	62,5703	45
28	62,0723	38
29	62,1013	37
30	62,378	35
Media	**62,3305129**	**36,3870968**
Mediana	**62,378**	

Se puede observar que los valores obtenidos son bastante mejores que los anteriores, si nos fijamos más detenidamente, ahora los valores del Hipervolumen se han estabilizados, es decir, casi no varían muchos entre ellos y se mueven entre un rango de valores bastante acotado. El hipervolumen ha subido y la cantidad de resultados que no son dominados han aumentado, esto nos indica que es un frente de pareto con bastantes soluciones, bien definido y bastante nítido.

6.1.6 Generar una población más acorde al problema

En el siguiente ajuste del algoritmo, nos centraremos en la generación de una población mejor desde un principio. Se venía explicando que la población era generada de manera aleatoria y estoy conllevaba a que hubiera diversidad en la población en cada ejecución del algoritmo. Ahora se ha cambiado por una concienciación del problema, es decir, se va a generar un a población acorde al problema que se le está planteando.

Lo que está ocurriendo es que se acota la cantidad de requisitos que son viables con respecto a la población de la que se dispone, esto quiere decir que se activarán los mejores requisitos que son calculados a priori, no se podrán activar todos los requisitos ya que se irán añadiendo requisitos a medida que vamos generando individuos, una vez que nos quedemos sin individuos que adaptar a nuestro problema, el resto de la población será creada de la misma manera que ha sido creada anteriormente.

Todo estará balanceado, ya que habrá individuos con pocos requisitos pero estos son elegidos de manera óptima, también dispondremos de individuos que poseen bastantes recursos, pero estos serán seleccionados de manera óptima y por último tendremos individuos de la población con muchos requisitos y con pocos requisitos pero generados de manera aleatoria. Esto dará la posibilidad de que las mutaciones sean más óptimas, ya que disponemos de mejores individuos a la hora de proceder a mutar la población y buscar una solución más óptima.

La solución generada es la siguiente:

Tabla 7: resultados de la modificación generación de población con acorde al problema.

0	61,4452	9
1	62,3332	36
2	61,5638	39
3	62,291	36
4	61,3728	38
5	62,3622	34
6	61,909	39
7	61,6033	35
8	61,9669	39
9	61,6692	41
10	61,9327	34
11	61,8312	41
12	61,7114	37
13	62,0658	36
14	61,9379	40
15	61,9801	42
16	61,7193	38
17	62,3872	30
18	61,8945	36
19	62,0236	37
20	61,6033	40
21	61,9577	39
22	61,4228	39
23	61,3912	41
24	61,5506	32
25	62,0473	39
26	62,2581	40
27	62,2238	40
28	62,5295	36
29	61,8444	37
30	61,2423	36
Media	**61,87326774**	**37,61290323**
Mediana	**61,909**	

Los resultados obtenidos esta vez, son un poco peor, pero hemos aumentado la tasa de valores no dominados.

6.1.7 Elección de modificación en el individuo

La modificación que se va a explicar a continuación, es lo que el propio título del apartado dice, pero se va a explicar qué va a ocurrir en este apartado con más detalle.

La elección de mejora en el individuo, consta de tres posibilidades: la eliminación de un requisito en el individuo, la sustitución de un requisito por otro y la inclusión de un requisito en el individuo. La modificación del requisito seguirá siendo la que se implementa en el apartado 6.1.6, la eliminación del requisito se regirá según la calidad de requisito a eliminar, es decir que sabemos cuáles son los mejores y peores a eliminar. El criterio que sigue la inclusión de un requisito se rige por el mismo criterio que en la eliminación de un requisito.

Para que se elija una de las tres opciones, tenemos que tener en cuenta la anterior modificación, es decir, se va a tener constancia de la población que se genera, dicho de otra manera, vamos a saber qué características va a tener cada individuo.

Ya que vamos a tener constancia de cada individuo vamos a saber de la cantidad de requisitos de los que dispone, por tanto si dispone pocos o de ninguno, elegirá aleatoriamente si decide entre o añadir un requisito o modifica un requisito. Si se disponen de muchos requisitos activados, se elegirá entre eliminar un requisito o modificar un requisito, normalmente se elegirá eliminar un requisito ya que al tener seleccionados muchos requisitos lo que se busca es quitar rebajar el esfuerzo, por último si disponemos de una carga intermedia de requisitos, se elegirá aleatoriamente entre las tres opciones.

Tabla 8: resultados de la modificación elección de modificación en un individuo.

0	61,4242	32
1	60,8827	39
2	61,2107	35
3	61,6442	35
4	61,4637	31
5	61,5216	40
6	60,8432	36
7	61,2292	31
8	61,4769	38
9	61,2463	34
10	61,88	35
11	60,7259	36
12	61,5599	34
13	61,4479	32
14	61,3227	35
15	61,7338	35
16	61,1225	37
17	61,5678	36
18	60,9038	33
19	61,5296	34
20	61,8668	32
21	61,8708	38
22	61,191	37
23	61,299	35
24	61,4057	35
25	60,7009	36
26	61,4479	39
27	61,5322	36
28	61,1804	33
29	61,3728	35
30	61,3491	34
Media	**61,35332903**	**35,09677419**
Mediana	**61,4057**	

La modificación que hemos hecho esta vez, ha dado como resultado un hipervolumen y una cantidad de resultados no dominados un poco peor que las anteriores modificaciones, pero podemos sacar algo positivo con esta modificación y es que los resultados que nos salen son bastantes más uniformes que los anteriores, es decir, el frente de pareto del que disponemos es bastante más uniforme que los anteriores.

6.1.8 Frecuencia de escritura en fichero

En esta modificación hemos tenido en cuenta la cantidad de veces que escribíamos en el fichero, esto ocurría solamente cuando un individuo era generado aleatoriamente y cuando terminábamos de mejorar la población, es decir, guardábamos la población, ahora es diferente, ahora se hace después de que un individuo es modificado de cualquiera de las tres manera, como ocurría en el apartado 6.1.6 por tanto en vez de disponer de unos 10000 individuos por cada prueba realizada en el algoritmo, ahora obtenemos el doble de individuos, también el tiempo de ejecución ha aumentado un poco, pero no es un problema, ya que nuestro programa en ejecutarse segundo y no nos importa esperar uno o dos segundos más.

Tabla 9: resultados de la modificación frecuencia de escritura en el fichero.

0	62,6349	39
1	62,8048	40
2	62,5624	42
3	62,681	40
4	62,8812	41
5	62,4926	41
6	62,6059	39
7	62,7482	36
8	62,3398	38
9	62,3674	40
10	62,7324	42
11	62,6928	39
12	62,8483	38
13	62,6296	37
14	62,5361	40
15	62,4965	40
16	62,7824	39
17	62,6296	37
18	62,4228	42
19	62,1303	42
20	62,677	37
21	62,4215	41
22	62,6388	38
23	62,5387	39
24	62,345	40
25	62,6204	43
26	62,6441	38
27	62,5124	44
28	62,6599	42
29	62,7877	38
30	62,4966	38

Media	**62,59229355**	**39,67741935**
Mediana	**62,6296**	

Podemos observar que la media general de Hipervolumen obtenido es mayor a las anteriores pruebas, además de la cantidad de soluciones no dominadas a aumentado, por tanto, esta modificación ha conllevado a la mejora general de los resultados de nuestro proyecto.

6.1.9 Ajustes en la configuración de charcos

A continuación se va a divisar las pruebas realizadas con un número diferentes de charcos y un número diferente de evaluaciones que ocurren en un charco. Por tanto cambiaremos la cantidad de charcos donde se reparten las ranas y la cantidad de evaluaciones que hacemos en ese charco.

La primera prueba realizada consta en ver el número de charcos que más le conviene a nuestro proyecto, por tanto, probaremos con diferentes números de charcos para ver cuál es mejor.

Tabla 10: cambiando los diferentes números de charcos.

Número de charcos	*Media HV*	*Media de resultados*
2	62.5842193548387	40.5806451612903
5	62.6463096774194	39.8064516129032
8	62.5811129032258	39.3870967741935
10	62.5948516129032	40.3225806451613
20	62.6031322580645	39.3548387096774

Como vemos, la cantidad de charcos que mejor conviene a nuestro proyecto es la de 5, ya que es la que más sobresale del resto de las otras pruebas. También se puede observar que la cantidad de resultados no dominados ha aumentado aunque sea poco, teniendo en cuenta que realizábamos las pruebas con 4 charcos hasta el momento, era por tener un número divisible.

La siguiente prueba a realizar consta de ver la cantidad de evaluaciones que se deben de realizar en un charco para que este mejore la población que hay en su interior. Se probarán diferentes evaluaciones y poder ver casos extremos y cómo afectan esos casos a nuestra población.

Tabla 11: resultados de las diferentes evaluaciones por charco.

Evaluaciones por charco	***Media HV***	***Media de resultados***
1	62.7385838709677	43.7741935483871
2	62.6317774193548	39.741935483871
5	62.609135483871	40.1290322580645
10	62.6300032258065	39.9677419354839
50	62.5763612903226	40.2903225806452
125	62.5891870967742	39.9677419354839
250	62.6248612903226	39.5483870967742
500	62.5999516129032	39.741935483871
1000	62.6094709677419	39.9354838709677
2000	62.5724032258065	39.6451612903226

Es curioso que los mejores resultados sean los casos extremos, por ello se explicaba antes de ver la tabla, es comprensible que el máximo valor que obtenemos sea con una sola evaluación, ya que se puede comprender que a menos que se modifique la población mejores resultados de, puesto que en el apartado 6.1.5 esta prepara la población para afrontar el problema, es decir que crea una buena población y a medida que la modifiquemos, podemos perder calidad en la población.

6.2 Resultados obtenidos

Los resultados que hemos obtenido han sido con todas las modificaciones anteriores implementadas y con la última modificación que vamos a explicar a continuación.

La modificación que se va a implementar es algo que de ser añadido al proyecto de forma necesaria y es añadir una de las restricciones que le falta al proyecto, es la restricción del esfuerzo ***Lc***, se puede divisar mejor en el apartado 4.7.1.

Ya que es nuestra última modificación, son los resultados finales que hemos obtenido de este proyecto, además del primer data set que se ha implementado en el proyecto, hemos implementado otro data set para hacer diferentes pruebas

Effort		r_1 16	r_2 19	r_3 16	r_4 7	r_5 19	r_6 15	r_7 8	r_8 10	r_9 6	r_{10} 18	r_{11} 15	r_{12} 12	r_{13} 16	r_{14} 20	r_{15} 9	r_{16} 4	r_{17} 16	r_{18} 2	r_{19} 9	r_{20} 3
Priority level	cl_1	1	2	1	1	2	3	3	1	1	3	1	1	3	2	3	2	2	3	1	3
	cl_2	3	2	1	2	1	2	1	2	2	1	2	3	3	2	1	3	2	3	3	1
	cl_3	1	1	1	2	1	1	1	3	2	2	3	3	3	1	3	1	2	2	3	3
	cl_4	3	2	2	1	3	1	3	2	3	2	3	2	1	3	2	3	2	1	3	3
	cl_5	1	2	3	1	3	1	2	3	1	1	2	2	3	1	2	1	1	1	1	3
Effort		r_{21} 2	r_{22} 10	r_{23} 4	r_{24} 2	r_{25} 7	r_{26} 15	r_{27} 8	r_{28} 20	r_{29} 9	r_{30} 11	r_{31} 5	r_{32} 1	r_{33} 17	r_{34} 6	r_{35} 2	r_{36} 16	r_{37} 8	r_{38} 12	r_{39} 18	r_{40} 5
Priority level	cl_1	2	1	1	1	3	3	3	3	1	2	2	3	2	1	2	2	1	3	3	2
	cl_2	3	3	3	2	3	1	2	2	3	3	1	3	2	2	1	2	3	2	3	3
	cl_3	2	1	2	3	2	3	3	1	3	3	3	2	1	2	2	1	1	3	1	2
	cl_4	1	1	1	2	3	3	2	1	1	1	1	2	2	2	3	2	2	3	1	1
	cl_5	1	1	3	3	3	2	2	3	2	3	1	1	3	3	2	2	1	1	2	1
Effort		r_{41} 6	r_{42} 14	r_{43} 15	r_{44} 20	r_{45} 14	r_{46} 9	r_{47} 16	r_{48} 6	r_{49} 6	r_{50} 6	r_{51} 6	r_{52} 2	r_{53} 17	r_{54} 8	r_{55} 1	r_{56} 3	r_{57} 14	r_{58} 16	r_{59} 18	r_{60} 7
Priority level	cl_1	2	2	3	1	1	1	2	2	3	3	3	3	1	3	2	1	3	1	3	1
	cl_2	3	3	1	1	3	2	2	2	1	3	3	3	1	2	2	3	3	2	1	1
	cl_3	1	3	1	3	3	3	3	1	3	2	3	1	2	3	2	3	2	1	2	3
	cl_4	3	1	1	3	1	2	1	1	3	2	2	1	3	2	1	3	3	1	2	3
	cl_5	3	1	1	2	1	2	3	3	2	2	1	3	3	2	3	1	2	1	3	2
Effort		r_{61} 10	r_{62} 7	r_{63} 16	r_{64} 19	r_{65} 17	r_{66} 15	r_{67} 11	r_{68} 8	r_{69} 20	r_{70} 1	r_{71} 5	r_{72} 8	r_{73} 3	r_{74} 15	r_{75} 4	r_{76} 20	r_{77} 10	r_{78} 20	r_{79} 3	r_{80} 20
Priority level	cl_1	2	2	3	3	1	3	1	3	2	3	1	3	2	3	1	1	2	3	3	1
	cl_2	1	3	2	3	1	2	1	2	3	1	1	3	1	3	2	1	3	3	1	2
	cl_3	1	1	2	3	3	1	3	3	3	1	3	1	3	1	1	2	3	3	1	2
	cl_4	2	2	3	3	3	1	2	1	2	1	2	3	3	2	2	2	1	3	3	1
	cl_5	2	2	1	2	1	3	2	1	2	1	2	2	3	2	1	3	2	3	1	3
Effort		r_{81} 10	r_{82} 16	r_{83} 19	r_{84} 3	r_{85} 12	r_{86} 16	r_{87} 15	r_{88} 1	r_{89} 6	r_{90} 7	r_{91} 15	r_{92} 18	r_{93} 4	r_{94} 7	r_{95} 2	r_{96} 7	r_{97} 8	r_{98} 7	r_{99} 7	r_{100} 3
Priority level	cl_1	2	1	3	1	2	2	2	1	3	2	2	3	1	1	1	2	1	3	1	1
	cl_2	1	2	1	2	2	1	3	2	2	2	3	2	2	3	2	2	1	3	1	1
	cl_3	1	2	3	2	3	1	2	2	3	3	3	3	2	1	1	2	3	3	2	3
	cl_4	3	1	2	2	2	1	1	1	3	1	1	3	3	1	2	1	2	3	1	3
	cl_5	3	2	1	2	2	2	2	1	3	3	3	1	1	3	1	3	3	3	3	3

Figura 32: Segunda tabla de configuración. Esta imagen ha sido sacada de José M. Chaves-González , Miguel A. Pérez-Toledano – 2015 – Differential evolution with Pareto tournament for the multi-objective next release problem.

Interactions	$r_2 \Rightarrow r_{24}$ $r_3 \Rightarrow r_{26}$ $r_3 \Rightarrow r_{27}$ $r_3 \Rightarrow r_{28}$ $r_3 \Rightarrow r_{29}$ $r_4 \Rightarrow r_5$ $r_6 \Rightarrow r_7$ $r_7 \Rightarrow r_{30}$ $r_{10} \Rightarrow r_{32}$ $r_{10} \Rightarrow r_{33}$ $r_{14} \Rightarrow r_{32}$ $r_{14} \Rightarrow r_{34}$ $r_{14} \Rightarrow r_{37}$ $r_{14} \Rightarrow r_{38}$ $r_{16} \Rightarrow$ $\Rightarrow r_{39}$ $r_{16} \Rightarrow r_{40}$ $r_{17} \Rightarrow r_{43}$ $r_{29} \Rightarrow r_{49}$ $r_{29} \Rightarrow r_{50}$ $r_{29} \Rightarrow r_{51}$ $r_{30} \Rightarrow r_{52}$ $r_{30} \Rightarrow r_{53}$ $r_{31} \Rightarrow r_{55}$ $r_{32} \Rightarrow r_{56}$ $r_{32} \Rightarrow r_{57}$ $r_{33} \Rightarrow r_{58}$ $r_{36} \Rightarrow r_{61}$ $r_{39} \Rightarrow r_{63}$ $r_{40} \Rightarrow r_{64}$ $r_{43} \Rightarrow r_{65}$ $r_{46} \Rightarrow r_{68}$ $r_{47} \Rightarrow r_{70}$ $r_{55} \Rightarrow r_{79}$ $r_{56} \Rightarrow r_{80}$ $r_{57} \Rightarrow r_{80}$ $r_{62} \Rightarrow r_{83}$ $r_{62} \Rightarrow r_{84}$ $r_{64} \Rightarrow r_{87}$ $r_{21} \oplus r_{22}$ $r_{32} \oplus r_{33}$ $r_{46} \oplus r_{47}$ $r_{65} \oplus r_{66}$				
Dataset 2	1	5	3	3	1

Figura 33: restricciones y pesos de los clientes. Esta imagen ha sido sacada de José M. Chaves-González, Miguel A. Pérez-Toledano – 2015 – Differential evolution with Pareto tournament for the multi-objective next release problem.

Este data set es mayor que el anterior, este va a disponer de 100 recursos y un total de 42 restricciones, la cantidad de clientes a complacer es la misma 5. Esto conlleva a que el tiempo de ejecución es mucho mayor que en las anteriores pruebas y que los resultados van a ser un poco inferiores, ya que el tamaño de la población a manejar es bastante grande.

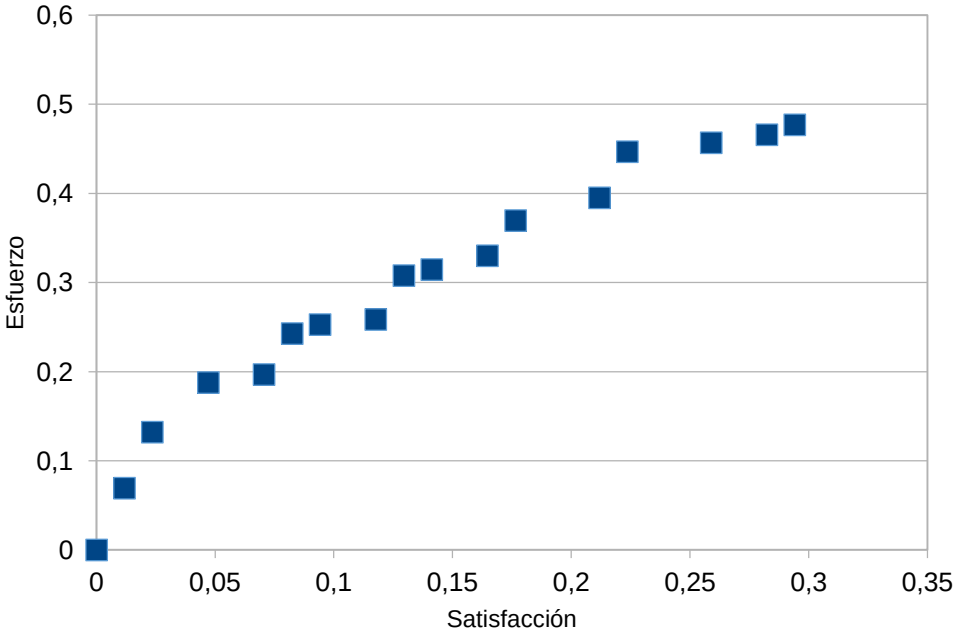
Veremos los resultados que hemos obtenido de data_set1 y data_set2 veremos sus frentes de pareto individualmente. Veremos los resultados que hemos obtenido con diferentes valores de Lc. Pasando por valores altos de restricción de esfuerzo como un 30% hasta llegar no tener límite de esfuerzo(100%). Se pasará por valores de intermedios además como 50% y 70% de límite de esfuerzo, así abarcaremos todos los casos posibles. No se hacen pruebas con un límite de esfuerzo del 0% porque no nos saldrían resultados.

Recordemos que el objetivo a cumplir en todas estas gráficas es llegar al punto (1,0), ese sería el punto idílico, pero no será posible llegar a ese punto ya que se deben de cumplir los objetivos. Por tanto lo que se intentará es aproximarnos a ese punto lo máximo posible.

Primero veremos los resultados que hemos obtenido a partir del data_set1, después se verán los datos obtenidos a partir del data_set2, por último compararemos los resultados, pero esto no se llevará a cabo hasta el próximo apartado.

Tabla 12: Resultados con el primer data set con esfuerzo al 30%.

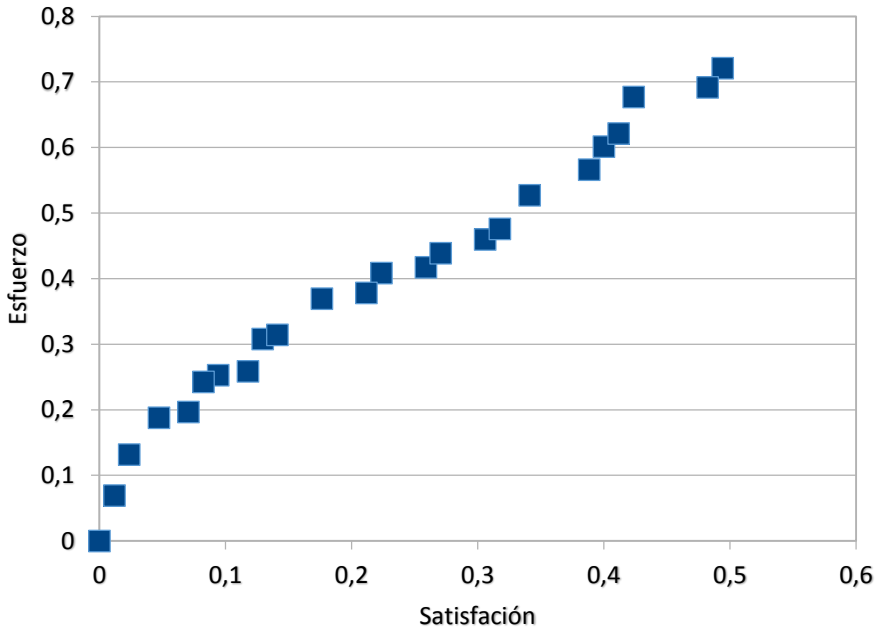
Lc 30%	*Hipervolumen*	*Resultados*
Media	42.7277838709677	16.9354838709677
Mediana	42.3872	17



Gráfica 1: Gráfica con data_set1 al 30%

Tabla 13: Resultados con el primer data set con esfuerzo al 50%.

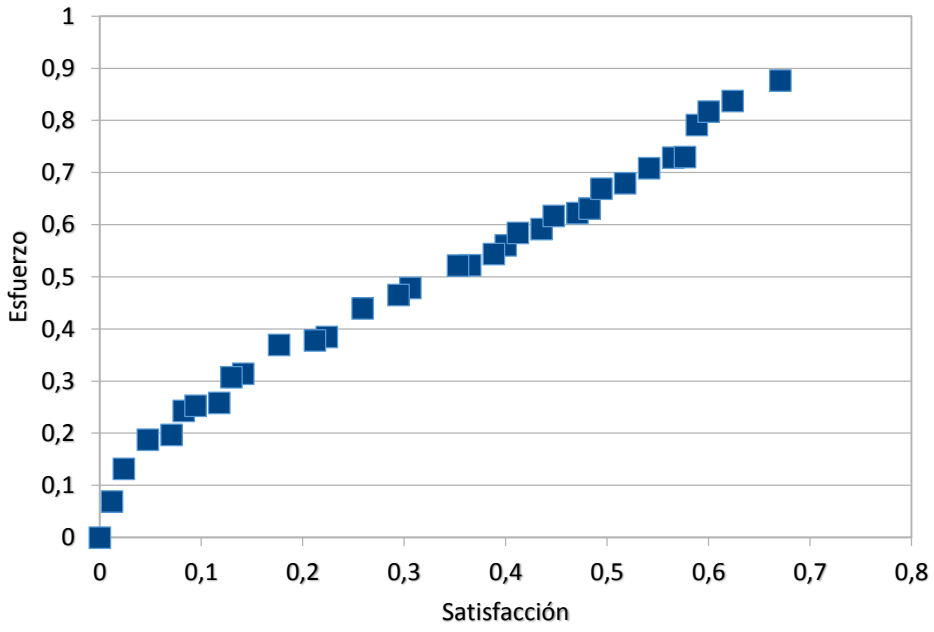
Lc 50%	*Hipervolumen*	*Resultados*
Media	56.2830580645161	22.5806451612903
Mediana	56.4891	24



Gráfica 2: Gráfica con data_set1 al 50%

Tabla 14: Resultados con el primer data set con esfuerzo al 70%.

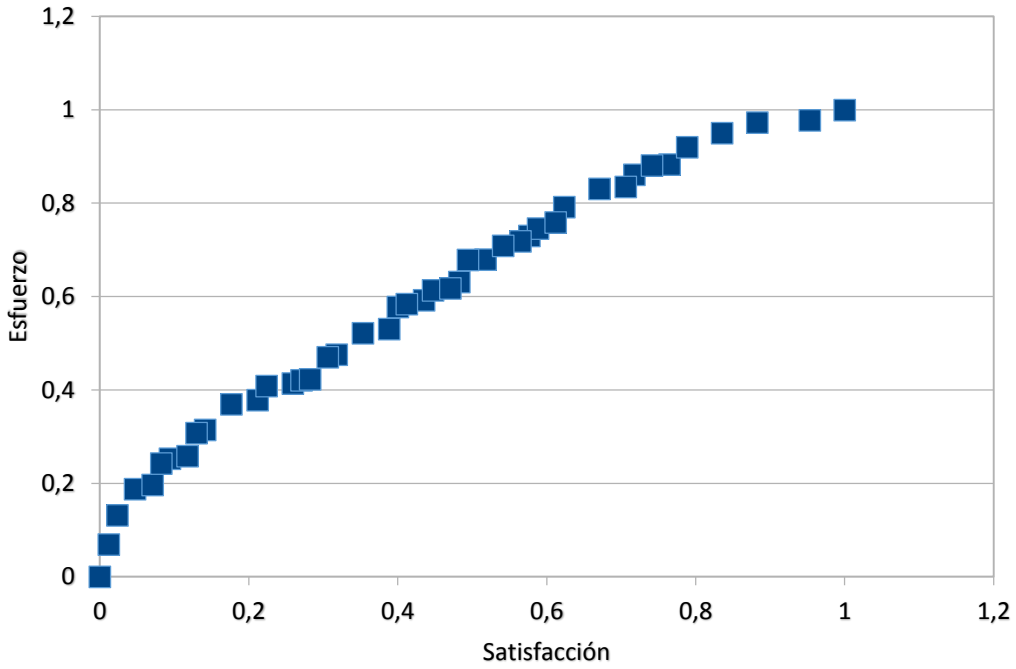
Lc 70%	***Hipervolumen***	***Resultados***
Media	61.5769096774194	32.0645161290323
Mediana	61.5757	35



Gráfica 3: Gráfica con data_set1 al 70%

Tabla 15: Resultados con el primer data set con esfuerzo al 100%.

Lc 100%	*Hipervolumen*	*Resultados*
Media	62.5758064516129	42.0967741935484
Mediana	62.5743	44



Gráfica 4: Gráfica con data_set1 al 100%

Haciendo un resumen, los resultados del data_set1 quedan de la siguiente manera:

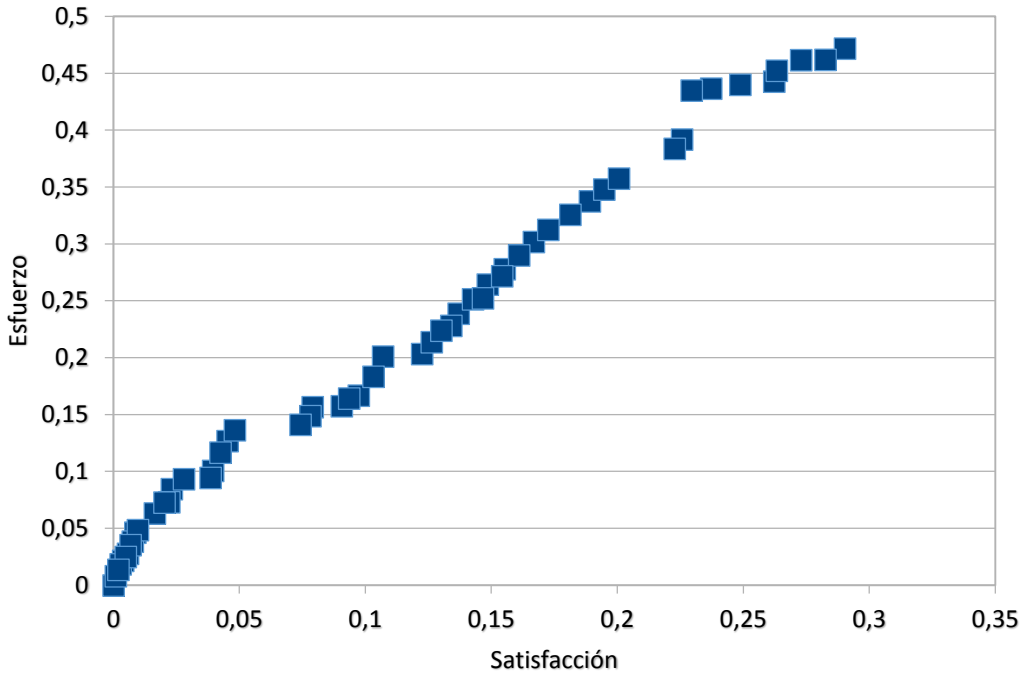
Tabla 16: Resumen de los resultados del primer data set.

LC	*Media Hipervolumen*	*Media Resultados*
30	41.0436483870968	54.9354838709678
50	56.2830580645161	22.5806451612903
70	61.5769096774194	32.0645161290323
100	62.5758064516129	42.0967741935484

A continuación se divisarán los resultados obtenidos a partir del data_set2, si nos percatamos, la cantidad de resultados que obtenemos es mayor ya que la cantidad de recursos e individuos a manejar ha aumentado.

Tabla 17: Resultados con el segundo data set con esfuerzo al 30%.

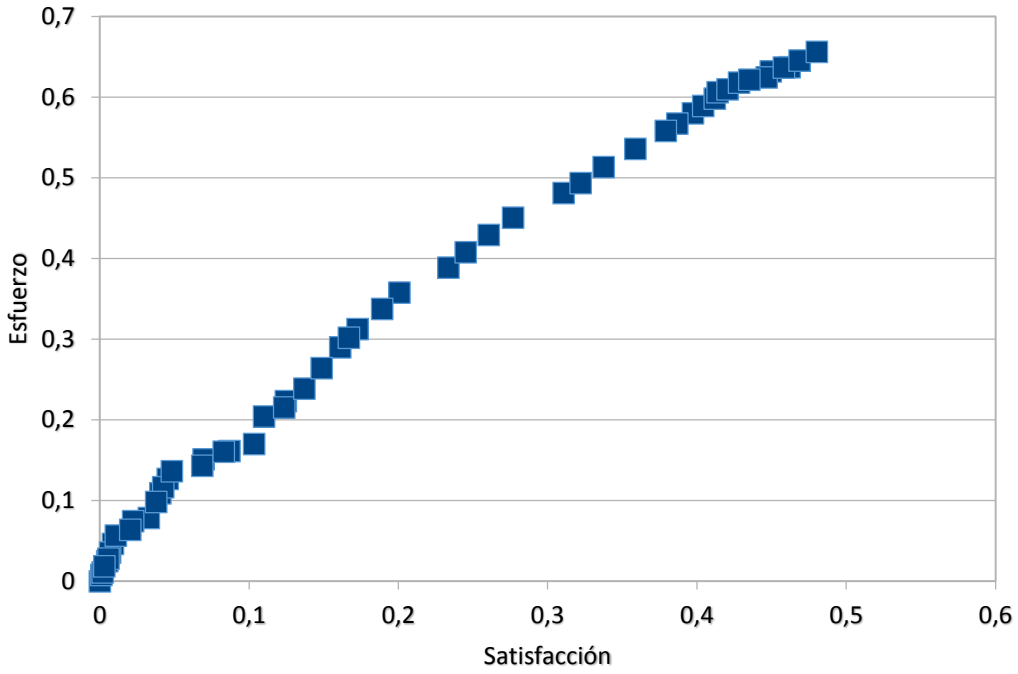
Lc 30%	***Hipervolumen***	***Resultados***
Media	41.0436483870968	54.9354838709678
Mediana	40.9932	56



Gráfica 5: Gráfica con data_set2 al 30%.

Tabla 18: Resultados con el segundo data set con esfuerzo al 50%.

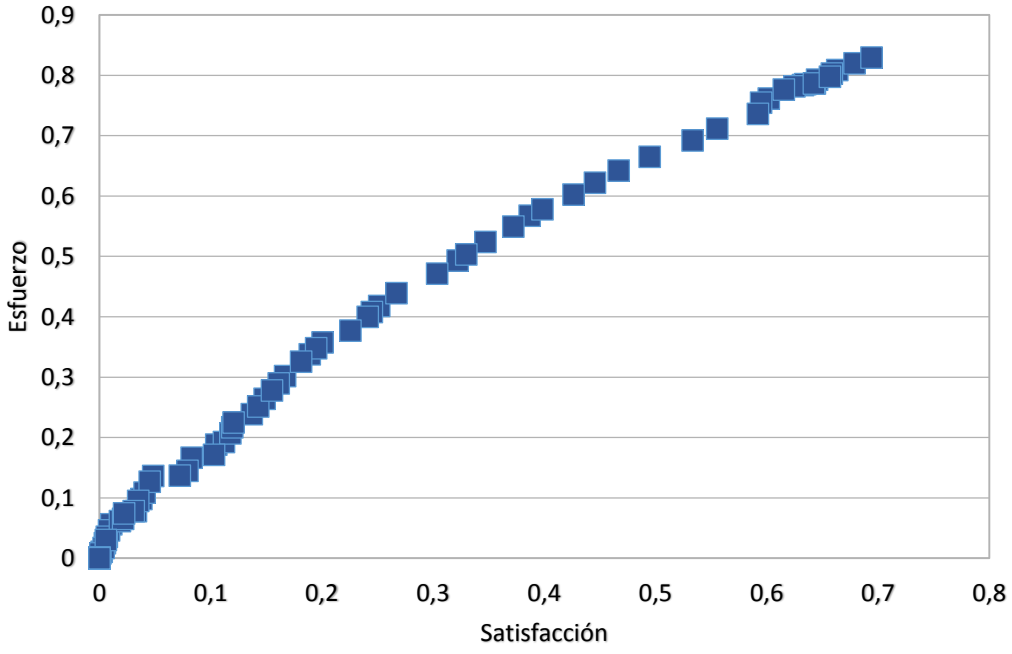
Lc 50%	***Hipervolumen***	***Resultados***
Media	52.1120193548387	66.3548387096774
Mediana	51.9351	55



Gráfica 6: Gráfica con data_set2 al 50%.

Tabla 19: Resultados con el segundo data set con esfuerzo al 70%.

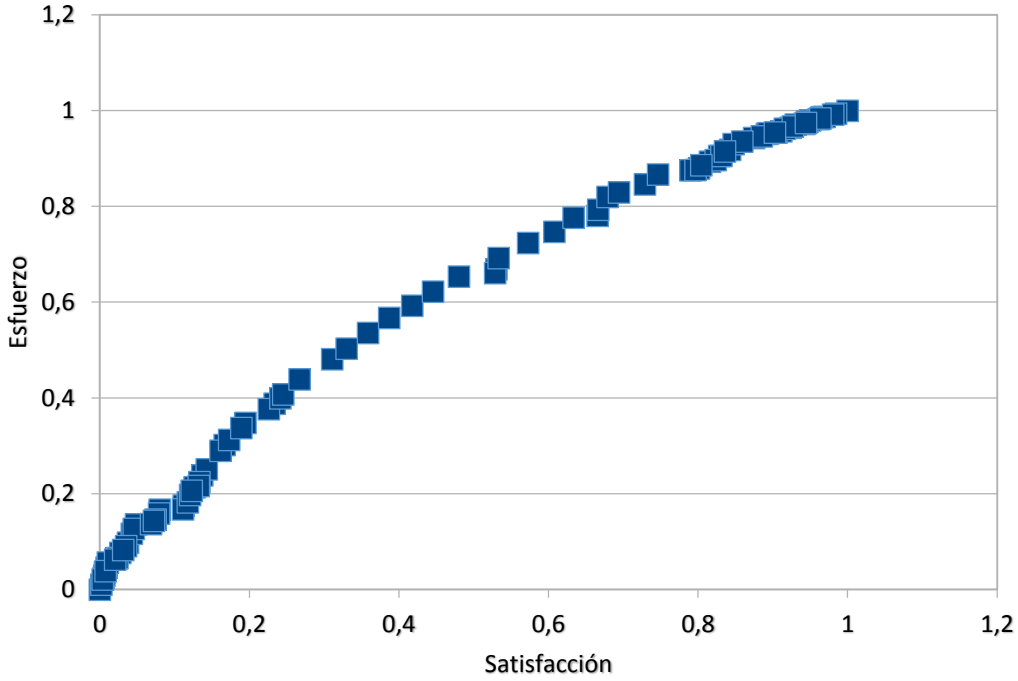
Lc 70%	***Hipervolumen***	***Resultados***
Media	58.7929806451613	76.5806451612903
Mediana	58.79	71



Gráfica 7: Gráfica con data_set2 al 70%.

Tabla 20: Resultados con el segundo data set con esfuerzo al 100%.

Lc 100%	*Hipervolumen*	*Resultados*
Media	60.9918516129032	94.9032258064516
Mediana	60.9811	99



Gráfica 8: Gráfica con data_set2 al 100%.

Aunando todos los resultados que hemos obtenido, podemos ver que con lo que se refiere al Hipervolumen los resultados son un poco menores que el data_set1. Pero la media de resultados obtenidos son bastante mayores, todo esto es porque la cantidad a manejar es bastante superior.

Tabla 21: Resumen de los resultados del segundo data set.

LC	*Media Hipervolumen*	*Media Resultados*
30	41.0436483870968	54.9354838709678
50	52.1120193548387	66.3548387096774
70	58.7929806451613	76.5806451612903
100	60.9918516129032	94.9032258064516

Generalizando, los resultados han ido aumentando a medida que hemos ido disminuyendo la restricción del esfuerzo, por tanto lo mejores resultados obtenidos en los data_set, son los que no disponen de límite de esfuerzo. Este caso es el que se toma de referencia para todas las pruebas, pero los resultados que nos deberían interesar, son los que llevan un límite, ya que estos son los que más se van a asemejar a un caso real.

6.3 Comparación de resultados

Compararemos nuestro algoritmo modificado MO-SFLA con otros algoritmos que se centran el mismo campo que el nuestro, principalmente lo compararemos con los resultados obtenidos del fichero *"José M.Chaves-González n, MiguelA.Pérez-Toledano,AmparoNavasa – 2015 – Teaching learning based optimization with Pareto tournament for the multiobjective software requirements selection"*. Nuestro objetivo es superar a MO-TLBO en todos los aspectos, el resto de algoritmos con los que comparamos, trabajan en el mismo campo de selección multiobjetivo de individuos. Todos los algoritmos que exponemos, han sido expuesto a las mismas condiciones, esto nos indica que ha se ha ejecutado el mismo número de veces y con los mismos archivos de configuración que se han mostrado previamente.

A continuación podremos divisar los diferentes resultados que se han obtenido con la primera configuración (data_set1):

Tabla 22: Resumen de los resultados del primer data set y comparación de resultados.

Esfuerzo	*MO-SFLA*	*MO-TLBO*	*ACO*	*NSGA-II*	*GRASP*
30	41.043%	42.981%	10.283%	9.015%	7.708%
50	56.283%	56.219%	23.912%	20.652%	19.114%
70	61.576%	62.837%	38.464%	32.157%	32.242%
Sin límite de esfuerzo	62.575%	66.562%	-	-	-

A continuación podremos divisar los diferentes resultados que se han obtenido con la segunda configuración (data_set2):

Tabla 22: Resumen de los resultados del segundo data set y comparación de resultados.

Esfuerzo	*MO-SFLA*	*MO-TLBO*	*ACO*	*NSGA-II*	*GRASP*
30	41.043%	43.182%	8.517%	7.920%	4.088%
50	52.112%	53.122%	19.159%	18.006%	15.454%
70	58.792%	59.992%	32.777%	31.710%	27.943%
Sin límite de esfuerzo	60.991%	64.126%	-	-	-

Nos podemos fijar que los resultados con 30% y 50% de esfuerzo superan al resto de los algoritmos, estos resultados son los que más se pueden asemejar a los problemas que se dan en la vida real. Pero solo esto es aplicable al data_set1. El resto de las pruebas, es decir la de 70% y sin límite de esfuerzo, nuestro algoritmo solo es superado por MO-TLBO.

Generalmente nuestro algoritmo da buenos resultados, tanto como para superar a todos los algoritmos en todas la pruebas, menos a MO-TLBO en las pruebas de 70% y sin límite de esfuerzo.

Con respecto al data_set2 MO-TLBO, supera a nuestro algoritmo en todas las pruebas, pero nuestro algoritmo supera al resto en todas las pruebas.

7. CONCLUSIÓN

En este apartado comentaremos las deducciones que hemos obtenido tanto como de nuestro proyecto y como del trabajo realizado, además comentaremos las soluciones. Finalizando este apartado incluiremos una propuesta de trabajo futuro.

7.1 Impresiones finales

Viendo los objetivos que se querían cumplir al principio en el apartado 3, podemos afirmar que se han cumplido todos, pero estos objetivos eran meras explicaciones previas antes de mostrar los resultados obtenidos.

Entender la optimización software era uno de los objetivos a cumplir, hoy en día este campo no está tan explorado como se debería, por ese motivo, he puesto bastante hincapié en cada concepto, ya que son conceptos nuevos con los que no estamos acostumbrados a trabajar. Creo que este campo de la informática puede ser explotado de una manera más óptima, pero para ello se necesita bastante trabajo por detrás.

El entendimiento de los algoritmos multiobjetivos era otra ardua tarea, cabe la posibilidad de que haya confusiones en esta parte, ya que ocurre lo mismo que en el párrafo anterior, son términos bastante recientes. Se ha intentado que quede todo lo más claro posible ya que casi todas las técnicas multiobjetivos han sido aplicadas a nuestro proyecto.

Otro de los objetivos a cumplir era la comprensión de los algoritmos basados en enjambres. Gracias a estos algoritmos hemos podido realizar el trabajo desde un punto de vista diferente, si no hubiéramos utilizado estos algoritmos, no se podría llevar a cabo el trabajo propuesto.

En general todos los conceptos aplicados en este proyecto son bastante complicados de entender, pero con tiempo es posible alcanzara a entender todos los conceptos, ya que no solo engloba conceptos de un solo campo de

la informática, si no que abarca varios campos. Una de las dificultades de este proyecto ha sido aunar estos conceptos en un solo proyecto, ya que entender cada parte por separado es complicado y si se unen partes, el grado de dificultad aumenta.

Otra de las dificultades ha sido cambiar la metodología de trabajo, ya que era necesario dar un punto de vista diferente con respecto al utilizado en el grado.

El objetivo principal de este proyecto era superar los resultados que comparábamos en el apartado anterior 6.3, esto no ha sido posible, pero esto no es negativo, ya que con trabajo hemos conseguido unos resultados bastantes parejos. Había buenas ideas en este proyecto, algunas mejores que otras, pero que uniéndolas, se esperaba que superásemos esa barrera que teníamos puesta. Podemos sacar en positivo que se ha superado parte de las pruebas.

7.2 Trabajo futuro

El trabajo que se propone es intentar mejorar ese algoritmo, ya que es posible un margen de mejora, el cual no se ha podido divisar, ya que las ideas que se proponían eran bastante buenas e innovadoras. Si hemos prestado atención a lo largo de todo este documento, se ha citado varias veces que cualquier modificación, por mínima que resultase en este proyecto, los cambios en los resultados eran bastante notorios. La clave de mejora sería hallar esa idea que mejorase los resultados de manera significativa.

8. BIBLIOGRAFÍA

José M. Chaves-González , Miguel A. Pérez-Toledano – 2015 – Differential evolution with Pareto tournament for the multi-objective next release problem.

José del Sagrado · Isabel M. del Águila ·Francisco J. Orellana – 2013 – Multi-objective ant colony optimization for requirements selection.

José M.Chaves-González n, MiguelA.Pérez-Toledano,AmparoNavasa – 2015 – Teaching learning based optimization with Pareto tournament for the multiobjective software requirements selection.

Jose M. Chaves-Gonzalez , Miguel A. Perez-Toledano, Amparo Navasa – 2015 – Software requirement optimization using a multiobjective swarm intelligence evolutionary algorithm.

Antonio Mauricio Pitangueira, RitaSuzanaP.Maciel,Márcio Barros – 2015 – Software requirements selection and prioritization using SBSE approaches: Aystematic review and mapping of the literature.

Philip Achimugu , Ali Selamat, Roliana Ibrahim, Mohd Naz'ri Mahrin – 2015 – A systematic literature review of software requirements prioritization research.

Yuanyuan Zhang, Mark Harman, Soo Ling Lim – 2013 – Empirical evaluation of search based requirements interaction management.

José Manuel García Nieto, Enrique Alba Torres y Gabriel Jesús Luque Polo – 2006 – Algoritmos Basados en Cúmulos de Partículas Para la Resolución de Problemas Complejos.

Tesis Sistema de generación eléctrica con pila de combustible de óxido sólido alimentado con residuos forestales y su optimización mediante algoritmos basados en nubes de partículas.

M. Méndez, B. Galván, D. Greiner, G. Winter – Algoritmos Evolutivos y Punto de Funcionamiento Aplicados a un Problema Real de Optimización Multiobjetivo en Diseño de Sistemas de Seguridad.

Pedro Manuel Ramos Rodríguez – 2017 – Herramienta para la validación estadística de metaheurísticas.

Álvaro García Sánchez – Técnicas Metaheurísticas.

David L. González – Álvarez – 2013 – Metaheurísticas, Optimización Multiobjetivo y Paralelismo para Descubrir Motifs en Secuencias de ADN.

José Francisco Chicano García – 2007 – Metaheurísticas e Ingeniería del Software.

EMAD ELBELTAGI, TAREK HEGAZY and DONALD GRIERSON – 2005 – A modified shuffled frog-leaping optimization algorithm: applications to project management.

Kalyanmoy Deb*, Associate Member, IEEE*, Amrit Pratap, Sameer Agarwal, and T. Meyarivan – 2002 – A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II.

Zitzler, E., Thiele, L., Laumanns, M., Fonseca, C., and Grunert da Fonseca, V (2003): Performance Assessment of Multiobjective Optimizers: An Analysis and Review. IEEE Transactions on Evolutionary Computation, 7(2), 117-132.

Vincent S Lai, Bo K Wong, Waiman cheung – 2001 – Group decision making in a multiple criteria environment: A cause the AHP in software selection – http://www.sciencedirect.com/science/srticle/pii/S0377221701000844

Función de calidad explicada – http://asq.org/quality-progress/2003/03/problem-solving/qfd-explicado.html

Diferencia entre método heurístico y optimización – https://augustosoberano.wordpress.com/2009/10/16/tarea-3-diferencia-entre-metodo-euristico-y-de-optimizacion/

Metaheurística – https://es.wikipedia.org/wiki/Metaheur%C3%ADstica

Programación evoulutiva – https://en.wikipedia.org/wiki/Evolutionary_programming

Nancy Mead – 2006 – Requeriments Prioritization Case Study Using AHP – https://www.us-cert.gov/bsi/articles/best-practices/requirements-engineering/requirements-prioritization-case-study-using-ahp