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

<Nombre y Apellidos del Autor>

<Convocatoria, Año>

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**

**Resumen**

**Introducción**

**cosas a entender**

**estructura de la documentación**

**Objetivos**

**Antecedentes**

**Requisitos software(teoría) – <**

**NRP**

**Técnicas de optimización – <**

**Metaheurísticas**

**Algoritmos basados en inteligencia de enjambres –**

**Algoritmo evolutivos –**

**Algoritmo de las ranas –**

**Optimización multiobjetivo –**

**Definición MO del problema de requisitos softwares –**

**Material Y Método**

**Hardware y Software usados**

**MO-SFLA**

**Resultados y discusión**

**Ajustes del algoritmo**

**Resultados obtenidos**

**Comparación de los resultados**

**Conclusión**

**Bibliografía**

**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.

**INTRODUCCIÓN**

Lo que vamos a ver en esta sección va a ser cómo va a estar estructurada nuestra documentación...(extender más tarde)

**Cosas a entender**

**Estructura de la documentación**

**Objetivos**

**Requisitos Softwares (Teoría)**

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 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.

(Imagen sobre la planificación de un proyecto software??)

Para que 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, esta 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.

(incluir referencias a los artículos de Miguel A.)

**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.

(No se si debo poner más teoría sobre los inicios de la búsqueda de requisitos)

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 Jerarquico** o PAJ, en inglés sería **AHP** (Analytical Hierarchy Process), donde los requisitos son calificado según un valor-costo.



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 clarida.

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:

 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 a cerca 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.

(NO SÍ SI DEBERÍA PONER UN EJEMPLO------PREGUNTAR)

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, si no 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(6).

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 si, 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 planteo por primera(Zang et al) vez el el problema de multi-objetivo 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.

(Creo que puedo acabar esta parte aquí – Preguntar)

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.

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 en encontrar una solución siguiendo ciertos criterios, esta solución hallada sera óptima pero no la más optima que se pueda encontrar, esto sería una contra, pero uno de los beneficios que tiene al 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 si 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.)

**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 eficiente. 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**.

ImagenXX:

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).

(Hablar sobre las metaheurísticas basadas en trayectorias si necesito más páginas)

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

**Algoritmos basados en población**

Estos algoritmos utilizan alguna clase de estructuración de los individuos de la población. 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) .



Imagen XX:

Por un lado, las metaheurísticas celulares (véase el esquema de la izquierda en la Imagen XX) 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 produce 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 Imagen XX), 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.

(REFRRENCIAR)

## Tipos de Algoritmos basados en población

(Dar introducción)

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.

**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. 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 Imagen XX 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.



Imagen XX:

Pasaremos a explicar el proceso evolutivo de la población en Imagen XX 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.

Fueron cuatro los primeros tipos de algoritmos evolutivos que surgieron. 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.

**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, 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 multi-agente 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).

Iagen XX:

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.



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.



El **Algoritmo de las Luciérnagas** (Firefly Algorithm – FA): este 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.



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 partíulas (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 fuerza presentes en la naturaleza.



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.

ALGORITMO DE LAS RANAS

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.



**Optimización multi-objetivo**

En este apartado como muy bien dice el título, vamos a aprender sobre la optimización multi-objetivo, 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 multi-objetivo no tiene mucho tempo. Veremos diversos términos que acompañan a la base de la OMO(otimización multi-objetivo), términos como el frente de pareto, más todas sus característica, 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.

**Primer apartado(nombrar más adelante)**

Como hemos explicado en el apartado de Optimización Software[...], si se pretendía hacer una OMO (optimización multi-objetivo), 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, si no que también se puede extrapolar a muchos otros campos ((como (no se me ocurren relaciones). Pero todos estos campo 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 multi-objetivo (MOP - Multiobjective Optimization Problem), debemos entender y dominiar 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 optimo, 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(explicadas en el apartado X).

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.

(COMPARAR OPTIMIZACIÓN MULTI-OBJETIVO CON MONO-OBJETIVO)

**PARETO(explicación)**

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.

En Pareto se dispone de un conjunto de soluciones , donde “n” es el número de objetivos, se dice que domina a, 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.

Cunado 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, sabre quién domina a quién.

Dado un vector , se dice que domina a otro vector si y sólo si:

V. Pareto se apoyo 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 se dice que es Pareto-óptima si y sólo si no existe otro vector *x* tal que domine a .

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**.



ImagenXX:

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 imagenX asdfSi 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.



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.

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 igual 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 sus 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 ⇒ rj]*** esto indica que un requisito ***ri*** no puede ser seleccionado si el requisito ***rj*** no ha sido seleccionado previamente.
2. Combinación o acoplamiento(⊕): ***[ri* ⊕ *rj]*** significa que un requisito ***ri*** debe ser incluido obligatoriamente si el requisito ***rj*** está seleccionado.
3. Exclusión (⊗): ***[ri ⊗ 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.

**Definición MO del problema de requisitos softwares**

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: *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:  *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:*

V =left(matrix{v\_11#v\_12#dotsaxis#v\_{1n}##v\_21#{} #{}#v\_{2n}##dotsvert#{}#{}#dotsvert##v\_{n1}#v\_{n2}#dotsaxis#v\_nn}right)

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

* W: *representa el peso de cada cliente, es decir, la prioridad que tiene ese cliente, cada Wi representa a un cliente.*
* E: *esfuerzo que conlleva hacer cada requisito, cada “ei” representa al esfuerzo de cada requisito.*
* S: *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:*

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: *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.*

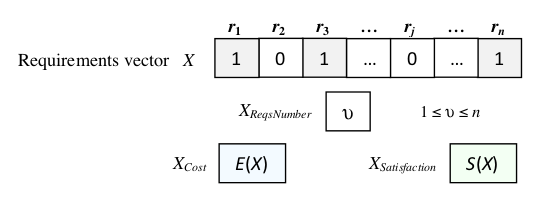
Unificando *X* con los interrelaciones de los requisitos, debemos tener en cuenta qué requisitos están presentes, pongamos un ejemplo, en la implicación, si ***ri*** pestá en X, es decir, si vale 1, **rj** debe estar también en X, lo mismo pasa con la combinación y con la exclusión.

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.

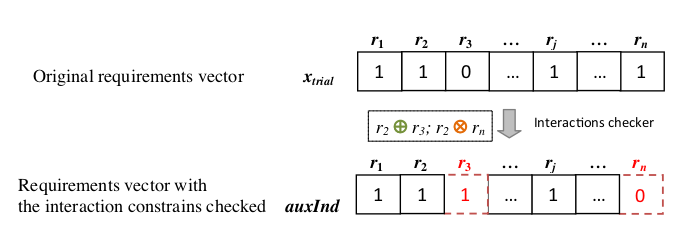
Nuestro objetivo es maximizar la satisfacción, es decir, que el conjunto de la suma de todas las satisfacciones salga lo mayor posible:

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:

**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)***.*

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 combinacion **(⊕)** y con la exclusión **(⊗)**.



(terminar de completar con el documento)