 

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**

**Requisitos software(teoría) – <**

**Técnicas de optimización**

**Optimización multiobjetivo –**

**Hardware y Software usados**

**Algoritmo de las ranas**

**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 entrar en materia.

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.

Nos hemos apoyado en un software externo que nos ayudaba a calcular el HV (Hipervolumen), es otro término que veremos más adelante.

El software se llama “hyp\_ind.c”, al cual le pasamos uno valores previos para que pueda tener un configuración a la hora de calcularnos el hipervolumen.

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 bastante competentes, están muy a la par con los resultados que comparamos, se podrían decir que los resultados son algo mejores que en un principio.\**

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

**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**

**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)

**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.

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**.



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.



El frente de Pareto es el con