ISSN 1870-4069

Aplicación de un algoritmo genético multiobjetivo para la replaneación de liberaciones en proyectos ágiles de software

Victor Hugo Escandon Bailon, Humberto Cervantes Maceda, Abel García Nájera

Universidad Autónoma Metropolitana
Ciudad de México, México
escvic@gmail.com, hcm@xanum.uam.mx,
agarcian@correo.cua.uam.mx

Resumen. El desarrollo de proyectos de software es un proceso que tiene una naturaleza dinámica, pues es común que en los proyectos de software los requerimientos y la composición del equipo presenten cambios. Estas modificaciones requieren de hacer cambios en la planeación del proyecto. A este proceso de adecuar la planeación para cumplir con la entrega a pesar de los eventos disruptivos que puedan ocurrir se le conoce como problema de replaneación de proyectos. Actualmente muchas organizaciones usan las metodologías ágiles, tales como Scrum, para desarrollar proyectos de software, en las cuales el software se desarrolla iterativamente y se entrega al cliente en incrementos llamados liberaciones. Para resolver el problema de replaneación en el contexto de las liberaciones se propone un modelo tomando en cuenta las características del desarrollo ágil usando Scrum. Para resolverlo, se propone un algoritmo genético multiobjetivo basado en NSGA-II. Se consideran los objetivos de tiempo, costo, estabilidad, desaprovechamiento de la capacidad de desarrollo y valor de la liberación. Todos los objetivos se describen con base en las definiciones de puntos de historia y velocidad de sprint usadas en el desarrollo ágil con Scrum. Los resultados muestran que el algoritmo presenta resultados coherentes.

Palabras clave: scrum, replaneación de proyectos, desarrollo de software dinámico, NSGA-II, optimización.

Application of a Multi-Objective Genetic Algorithm to the Release Replanning in Software Agile Projects

Abstract. Software project development is a process that has a dynamic nature, as it is common that in software projects the requirements and the team composition change. These modifications require making changes in the project planning. To the process of adapting the planning to comply with the delivery, despite the disruptive events that may occur, is known as the project rescheduling problem. Currently, many organizations use agile methodologies, such as Scrum, to develop software projects, in which the software is developed iteratively and it is delivered to the client in increments named releases. To solve the project replanning problem in the context of releases, a model is proposed taking into

account the characteristics of agile development using Scrum. To solve this problem, a multi-objective genetic algorithm based on NSGA-II is proposed. The objectives time, cost, stability, development capacity underuse and release value are considered. All objectives are described based on definitions of story points and velocity that are used in agile development with Scrum. The results show that the algorithm presents coherent results.

Keywords: scrum, project replanning problem, dynamic software development, NSGA-II, ptimization.

1. Introducción

En la actualidad muchas organizaciones han adoptado el desarrollo de proyectos de software con metodologías ágiles, particularmente Scrum, la cual tiene más de 20 años de desarrollo [1]. En estas metodologías, el software se desarrolla iterativamente y se entrega al cliente en incrementos llamados *liberaciones*. En las liberaciones se busca desarrollar funcionalidad del sistema que aporte valor al negocio del cliente de forma rápida. Al inicio del proyecto se realiza la planeación de una o más liberaciones. Durante la ejecución del proyecto, es común que se presenten eventos que afectan al plan, a los cuales llamaremos *eventos disruptivos*. Un ejemplo de un evento disruptivo es que un empleado se vaya o que se agregue un nuevo requerimiento (llamados historias de usuario en el contexto ágil). A la acción de ajustar o actualizar el plan original después de la ocurrencia de un evento disruptivo se le conoce como *replaneación*. Ya que el desarrollo de software es costoso y generalmente tiene fechas de entrega definidas, los administradores de proyectos deben realizar de inmediato una replaneación que minimice los impactos económicos y operativos, y que cumpla con las fechas de entrega que ya estaban definidas.

Además, existen mínimo otros tres objetivos importantes que se deben valorar en un proyecto real. Se desea que al realizar una replaneación ésta no difiera demasiado de la original, ya que se considera que la planeación inicial es la mejor opción para el desarrollo del proyecto. De igual manera se busca que las historias de usuario (HU) de mayor prioridad para la liberación se asignen a los primeros sprints y así desarrollar las HU más importantes primero. En el desarrollo de software los equipos son costosos y se debe usar de la mejor manera el tiempo de desarrollo de los empleados para evitar que este se desaproveche. Por lo tanto, al realizar una replaneación se podrían considerar cinco objetivos: tiempo, costo, estabilidad, valor de liberación y desaprovechamiento de la capacidad de desarrollo. Por lo anterior, el problema de replaneación es un problema de optimización multiobjetivo.

Este problema de replaneación se puede ver como una generalización del problema de asignación, el cual es bien conocido que pertenece a la clase NP-difícil. Además, algunos de los objetivos considerados están en conflicto, por ejemplo, si aumentamos el equipo de desarrollo buscando minimizar el tiempo de entrega, el proyecto será más costoso. Por esta razón, la solución que proponemos para este problema está basada en un algoritmo genético multiobjetivo, el cual considera a cada objetivo igualmente importante y busca dar como resultado un conjunto de propuestas de replaneación que sirvan como apoyo a los líderes de proyecto.

Los resultados obtenidos muestran que el algoritmo tarda un poco menos de 7 minutos para soluciones que proponen replaneaciones compuestas por 16 sprints, esto equivale a 240 días de proyecto. Muestran que, al aplicar un operador de reparación aumenta el indicador de calidad de hipervolumen en la población resultado. Esto mejora el desempeño del algoritmo, ya que el tiempo de ejecución es prácticamente el mismo.

Nuestro trabajo está estructurado de la siguiente manera: A continuación, en la Sección 2, se presenta la revisión bibliográfica. El problema de replaneación se describe en la Sección 3. En la Sección 4 se presenta la propuesta de solución al problema de replaneación. Después, en Sección 5, se muestran los resultados obtenidos y el análisis de los mismos. Finalmente, nuestras conclusiones se encuentran en la Sección 6.

2. Revisión bibliográfica

En esta sección presentaremos las ideas más importantes de algunos artículos, además de los enfocados en metodologías ágiles también tomaremos algunos que proponen soluciones al problema de replaneación.

De acuerdo a la revisión de la literatura hasta el momento, sólo Roque et al. [4] y Nigar [5] estudian el problema de replaneación en las metodologías ágiles, los cuales solo son introductorios y únicamente presentan las características del problema, por lo que no presentan una herramienta que sirva de apoyo a los administradores de proyectos de software en el contexto de las metodologías ágiles. En ellos usan heurísticas y se enfocan en las metodologías ágiles. En el primero [4], se propone un modelo sin replaneación por lo que presenta un modelo estático, es decir, sin cambios con el tiempo. Presentan una propuesta de solución al problema de planeación de proyectos, los autores proponen un algoritmo genético y sus objetivos son el tiempo y el costo del proyecto. Se enfocan en la fase de liberación, pero sin presentar eventos disruptivos y se realiza una planeación para cada sprint, pero no considera la replaneación a partir de un plan existente. En su modelo se busca realizar la asignación de empleados a las tareas. El segundo [5], sólo propone características para el modelo de planeación/replaneación de proyectos en metodologías ágiles ya que no presenta un algoritmo ni modelo para el problema. En este trabajo sólo se mencionan los objetivos que buscarán usar, los cuales son tiempo, costo, robustez, estabilidad y la fragmentación de HU (impacto de que una HU se retrase). Al ser sólo una propuesta, no tiene más detalle y sólo menciona eventos disruptivos generales.

La mayoría de artículos encontrados toman características de los modelos desarrollados para las metodologías tradicionales, tales como la asignación de empleados a las tareas o que la planeación se realiza solo al inicio del proyecto. Jahr [3] considera la planeación de proyectos en un contexto ágil, propone un método de programación entera para resolverlo y sólo toma como objetivo el tiempo que tarda el desarrollo. No encontramos replaneación, pero presenta asignación de los empleados a las tareas y no considera una estimación adecuada para las HU.

Los artículos en donde se modela el problema de replaneación hacen énfasis en que el desarrollo de software es un sistema dinámico. Los cambios se presentan durante la fase de ejecución debido a que ocurren eventos disruptivos. Pero, sin contar los trabajos de Jahr [3], Roque et al. [4] y Nigar [5], en todos los casos se enfocan a las metodologías tradicionales. Presentan propuestas que buscan resolver el problema de planeación de

proyectos de software, pero ahora en un entorno dinámico. En el trabajo de Ge [6] podemos encontrar un modelo para el problema de replaneación, como propuesta de solución implementa un algoritmo genético que busca minimizar el tiempo y costo de proyecto. Se introduce la estabilidad como objetivo del problema y los eventos disruptivos que considera son dos: llegada de nuevas tareas y el movimiento de los empleados (cuando llegan o se van).

Como antes se mencionó, el problema de replaneación es actual y en los últimos dos años podemos resaltar tres documentos dirigidos a las metodologías tradicionales, comenzando con el de Ge y Xu [7], en el cual los autores buscan minimizar el tiempo, costo y estabilidad del proyecto. Los eventos disruptivos que disparan una replaneación son: la llegada o retiro de empleados al proyecto, así como cuando llegan tareas que no se tenían en la planeación original. Los autores buscan modelar la productividad del equipo y cómo afecta ésta en el desarrollo de sus habilidades. Como en los casos anteriores la propuesta de solución está basada en un algoritmo genético.

En el estudio de Shen et al. [8] encontramos por primera vez una función multiobjetivo propuesta en el modelo, de esta manera cada objetivo se calcula de manera independiente. La propuesta presenta como solución a un conjunto de posibles propuestas de replaneación. Los objetivos que se buscan optimizar son: duración del proyecto, tiempo de desarrollo, estabilidad y robustez. Ellos consideran la llegada de nuevas tareas, el retiro de un empleado del proyecto y cuando un empleado se incorpora, como eventos disruptivos. Para validar su trabajo utilizaron tres proyectos del mundo real y los compararon con las propuestas que su algoritmo daba como solución. Como en el caso anterior, Shen et al. [9] también presenta una función multiobjetivo la cual se compone de: la duración del proyecto, el costo del desarrollo, la estabilidad entre planeaciones, la robustez para buscar un plan tolerante a eventos e introducen como objetivo la satisfacción de los empleados al ser asignados a tareas de su agrado. Como propuesta de solución usan un algoritmo compuesto de dos heurísticas. Como solución global proponen un algoritmo genético el cual da como resultado una propuesta de planeación, después, tratan de mejorar a esta propuesta aplicando un algoritmo AMDE (Angle Modulated Differential Evolution). Mencionan que para validar su algoritmo se compararon con proyectos reales, además de casos creados para prueba.

Nos damos cuenta que además de los objetivos tiempo, costo y estabilidad, algunos de los artículos consideran la robustez. Este objetivo se define como qué tan propensa es la replaneación a retrasarse en las fechas de entrega y costo cuando ocurre algún evento disruptivo. Es decir, la robustez es la capacidad de la replaneación a hacer frente a pequeños cambios.

Para proponer un modelo más cercano a la realidad, en los trabajos de Xiao et al. [10], Chen et al. [11] y Ge et al. [7], los autores modelan la comunicación necesaria entre los empleados para desarrollar una tarea. Esta comunicación afecta al proyecto si muchos empleados están asignados a la misma tarea, porque gastan mucho tiempo en comunicarse. Por último, encontramos que en trabajos más actuales se modela la productividad y la curva de aprendizaje de los empleados, podemos ver que esta característica se presenta en Ge et al. [7], Roque et al. [4] y mejoran el modelo presentado en Ge [6].

Como podemos darnos cuenta, pocos artículos hablan del problema de replaneación y tan solo tres están enfocados en las metodologías ágiles. Ninguno presenta

características del desarrollo ágil usando Scrum, ya que no utilizan conceptos como velocidad de sprint y puntos de historia. Por lo anterior, no existe una herramienta que presente propuestas para resolver el problema de replaneación en las metodologías ágiles.

Nuestro trabajo busca proponer un modelo de replaneación de proyectos desarrollados con metodologías ágiles. El modelo que presentamos toma en cuenta cinco objetivos a optimizar: tiempo, costo, estabilidad, aprovechamiento de la capacidad de desarrollo y valor de liberación. Estos dos últimos objetivos son parte de nuestras aportaciones y se explican más adelante.

3. El problema de replaneación de liberaciones en proyectos ágiles de software

En el presente trabajo proponemos un modelo para el problema de planeación en las metodologías ágiles (Scrum), en especial nos enfocamos en la replaneación de liberaciones. Llamaremos a este, problema de replaneación de liberaciones en proyectos ágiles de software (RLPAS). Después de un evento disruptivo se debe ajustar la planeación lo antes posible y con el menor número de cambios, de modo que el costo total y el tiempo de finalización no se vean afectados o, en su defecto, que el incremento sea mínimo, ya que el incremento en el costo debe ser absorbido por la empresa que lo desarrolla y esto se traduce en pérdida.

Las liberaciones están enfocadas a entregar funcionalidad específica y marcan las fechas de entrega de los incrementos al sistema. Cada liberación se divide en una o más iteraciones, llamadas *sprints*, como podemos ver en el ejemplo de la Fig.1. En la planeación de liberación se seleccionan las historias de usuario necesarias para cumplir con el objetivo de cada liberación y se asignan a algún sprint para calendarizar el orden en que serán desarrolladas. Al realizar una replaneación, se considera la asignación de HU al sprint que mejor convenga al plan, tomando en cuenta la duración, prioridad y dependencias de éstas.

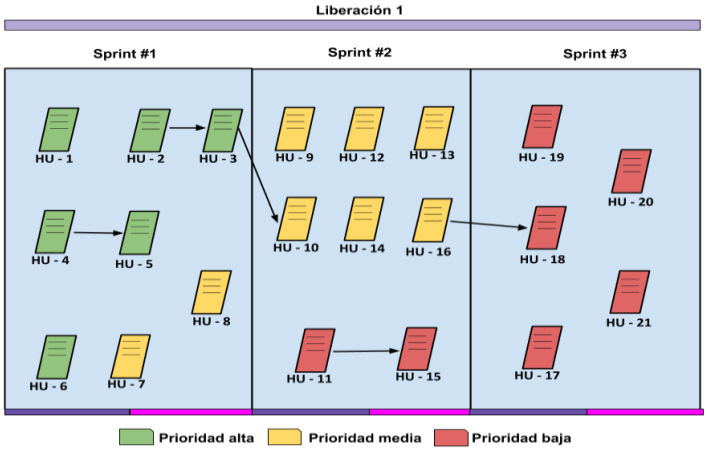

Fig. 1. Representación de un ejemplo de planeación de liberación, el cual consta de tres sprints.

La replaneación de liberaciones tiene características propias, las cuales, principalmente están orientadas a hacer más flexible y ágil la calendarización de las HU. Antes de presentar el modelo, es importante introducir algunos conceptos que son importantes en el contexto ágil, y más específicamente en Scrum.

Puntos de historia. En Scrum, la estimación del esfuerzo necesario para desarrollar las historias de usuario recae en el equipo de trabajo. Las HU se estiman con una unidad llamada *puntos de historia*, que representa el esfuerzo necesario para desarrollar una HU respecto a otra de referencia. Esta técnica se conoce como planning poker [2]. En este trabajo consideramos que el equipo ha hecho una estimación del esfuerzo de todas las HU antes de que se planee la liberación.

Velocidad de sprint. Por otro lado, la velocidad de sprint es una métrica histórica de la capacidad que ha mostrado tener el equipo para desarrollar de forma completa historias de usuario en un sprint.

Nuestro modelo contempla el tiempo extra que un equipo de desarrollo puede trabajar. Si un equipo trabaja horas extra puede tener más capacidad de desarrollo en un sprint (aumenta la velocidad de sprint) [5], pero el proyecto se volverá más costoso, pues es necesario pagar tiempo extra. En este trabajo consideramos un 22.5% de trabajo extra, en una jornada de 8 hrs este porcentaje representa 2 hrs extra[1].

Cuando se presenta un evento disruptivo y se tiene que hacer una replaneación, se intenta que estos eventos tengan el menor impacto sobre el proyecto, considerando los siguientes criterios.

Costo. Es el costo del equipo de trabajo más las horas extras que se necesiten en la replaneación. Para calcularlo se obtiene la velocidad de sprint regular y la velocidad de sprint considerando el trabajo extra.

Tiempo. Es la cantidad de sprints que son necesarios en la replaneación.

Estabilidad. Se refiere a las diferencias de asignación de historias de usuario a los sprints entre la planeación de liberación original y la que resulta de la replaneación. Se busca minimizar este objetivo.

Desaprovechamiento de la capacidad de desarrollo. Este objetivo mide la velocidad de sprint que la replaneación está "desaprovechando". Consideraremos como desaprovechamiento de igual manera que se use tiempo extra cuando aún hay tiempo regular disponible. En los proyectos se debe ocupar de la mejor manera el tiempo de desarrollo, ya que, como se mencionó antes, este es muy costoso. Una buena replaneación debe asegurar que en cada sprint se ocupe la cantidad máxima de tiempo disponible (velocidad de sprint). Al ajustar el plan después de un evento disruptivo, la nueva asignación de HU debe buscar la mejor combinación de acuerdo a la suma de los puntos de historia de cada sprint, para que la velocidad de sprint en cada uno sea ocupada, de preferencia, en su totalidad.

[1] El artículo 66 de la ley federal del trabajo en México, marca que una jornada extraordinaria de trabajo no puede exceder de 3 horas ni de 3 días a la semana. Lo que equivale aproximadamente a un 22.5% en una semana de trabajo de lunes a viernes.

Valor de liberación. Para que el modelo se ajuste adecuadamente, en una planeación o replaneación se deben desarrollar las HU con mayor prioridad en los primeros sprints. En una liberación buscamos entregar funcionalidad que aporte el mayor valor al negocio del cliente. Por ejemplo, en una tienda en línea las HU prioritarias son aquellas que permiten realizar una compra. Al realizar una replaneación después de un evento disruptivo, se deberían asignar las HU con mayor prioridad a los primeros sprints, con el fin de aumentar la probabilidad de tener el tiempo suficiente para poder desarrollarlas.

En nuestro trabajo, el problema de replaneación de liberaciones de proyectos ágiles de software se modela como un problema de optimización multiobjetivo, que busca minimizar el impacto de eventos disruptivos en todos los objetivos que se describieron anteriormente. A continuación, presentamos formalmente el modelo propuesto.

En un proyecto ágil de software usando Scrum, se tiene un conjunto $H = \{h_1, h_2, \dots, h_n\}$de n historias de usuario $h_i, \forall\, i \in \{1, \dots, n\}$, que se tienen que desarrollar. Cada historia de usuario h_i tiene una prioridad πh_i y ph_i puntos de historia estimados.

Definiremos un plan de liberación L, como el conjunto $L = \{S_1, S_2, \dots, S_m\}$de m sprints $S_i, \forall\, i \in \{1, \dots, m\}$. A su vez, cada sprint S_i es el conjunto $S_i = \{h_1^i, h_2^i, \dots, h_{n_i}^i\}$ de n_i historias de usuario $h_j^i \in H, \forall\, j \in \{1, \dots, n_i\}$, que se tienen que desarrollar en el sprint S_i. Así, los puntos de historia phS_i del sprint S_i se definen como:

$$phS_i = \sum_{j=1}^{n_i} ph_j. \tag{1}$$

Todos los sprints son conjuntos disjuntos, de tal forma que $S_1 \cup S_2 \cup \cdots \cup S_m = H = \{h_1, h_2, \dots, h_n\}$ contiene a las n historias de usuario h_j que se tienen que desarrollar en el plan de liberación L.

Para poder desarrollar las historias de usuario, se cuenta con un equipo de trabajo de k empleados y cada empleado $e_i, \forall\, i \in \{1, \dots, k\}$, aporta una velocidad de desarrollo ve_i a la capacidad de desarrollo del equipo. De esta forma, definimos la velocidad vS_i del sprint S_i como:

$$vS_i = \sum_{i=1}^{k} ve_i. \tag{2}$$

Habiendo definido formalmente el problema, ahora podemos plantear las cinco funciones objetivo que se consideran en este trabajo. Primero, el costo del proyecto se define como:

$$Costo = \sum_{i=1}^{m} (\, vS_i + veS_i), \tag{3}$$

en donde veS_i es la velocidad extra del sprint S_i: $veS_i = 0$ si $phS_i - vS_i \leq 0$ y $veS_i = phS_i - vS_i$ si $phS_i - vS_i > 0$.

El tiempo de entrega del proyecto es simplemente la cantidad de sprints considerados, es decir

$$Tiempo = m. \tag{4}$$

La estabilidad de la replaneación se define como:

$$Estabilidad = \frac{1}{(m-1)\cdot n}\sum_{i=1}^{n} |\, sprint(i,t) - sprint(i,t-1)|, \quad (5)$$

en donde $sprint(i,t)$ indica el sprint al cual está asignada la historia de usuario h_i al tiempo t, por lo que $sprint(i,t) \in \{1, \dots, m\}$. El factor $1/(m-1)\cdot n$ se incluye para obtener un valor normalizado.

El desaprovechamiento de la capacidad de desarrollo se define como:

$$Desaprovechamiento = \sum_{i=1}^{m} |\, vS_i - phS_i \,|. \quad (6)$$

Finalmente, el valor de liberación se define como:

$$Valor = \frac{1}{a}\sum_{i=1}^{m-1}\sum_{j=i+1}^{m}\sum_{k\in S_j}\sum_{l\in S_j}(\pi h_k - \pi h_l)(j-i), \quad (7)$$

en donde $a = 2(m-1)n^2$. Esta función objetivo da como resultado un número entre -1 y 1, en donde 1 significa que la planeación tiene el mejor valor de liberación y -1 el peor valor de liberación.

En los experimentos que se presentarán más adelante, se consideran dos tipos de eventos disruptivos: se agrega una nueva HU y se va un empleado. Sin embargo, existen otros tipos de eventos disruptivos, por ejemplo, no se termina o se elimina una HU, llega un empleado y un empleado es sustituido.

4. Algoritmo genético multiobjetivo para resolver el RLPAS

Con nuestra solución buscamos proveer a los líderes de proyecto un conjunto de propuestas de replaneación en unos pocos minutos. Ya que en proyectos reales un experto generalmente presenta una propuesta de replaneación como mínimo en 180 minutos [7], sin garantizar que sea una buena solución. Al modelar el problema de replaneación como uno multiobjetivo, el resultado es, idealmente, un conjunto de soluciones que presentan los mejores valores encontrados para cada uno de los objetivos y soluciones que presentan una compensación entre ellos.

Nuestro algoritmo calcula la aptitud de cada una de las propuestas de replaneación implementando el concepto de dominación. Se dice que una solución x *domina* a otra solución *y* si x es al menos tan buena como *y* para todos los objetivos y es estrictamente mejor en al menos uno de ellos. Las soluciones con mejor aptitud son aquellas que no son dominadas por ninguna otra, a esto se le conoce como *no dominancia*.

Para conocer el desempeño de los algoritmos multiobjetivo se usan indicadores de calidad, por ejemplo, el de hipervolumen, que dan como resultado un valor que representa el desempeño de un algoritmo respecto a otro. El indicador de hipervolumen (HV) [13] mide el espacio que cubren las soluciones no dominadas respecto a un punto de referencia y esto da cierta información de la convergencia y la diversidad de los individuos en la población. Por lo tanto, una población es mejor respecto a otra si su HV es mayor.

Existen varios métodos para resolver problemas de optimización multiobjetivo: exactos, heurísticos y metaheurísticos. Por las características del problema bajo estudio, hemos empleado un método metaheurístico, en particular, un algoritmo genético (AG) basado en NSGA-II [14].

Los algoritmos genéticos se basan en los principios de evolución mediante la selección natural. A grandes rasgos, un algoritmo genético considera una función a optimizar, una representación de la solución llamada cromosoma, operadores genéticos y una función que mide la aptitud del cromosoma. La búsqueda que realiza un AG debe tener un balance entre explotar y explorar el espacio de búsqueda. Un algoritmo para un AG consiste en:

1. Representación de la solución como un cromosoma.
2. Generación de una población.
3. Aplicación repetida de los operadores genéticos.

Para nuestra propuesta de solución al problema de replaneación se implementó NSGA-II [14], el cual es un algoritmo genético multiobjetivo que presenta elitismo y conserva la diversidad de la población. Cada individuo (cromosoma) de la población representa una replaneación. NSGA-II implementa una clasificación rápida basada en la no dominancia, en esta se comparan todos los individuos de la población y se les asigna un nivel. La aptitud de cada individuo depende de su nivel de no dominancia, donde cero representa la mejor aptitud. La segunda característica de NSGA-II es la distancia de amontonamiento de un individuo. Para calcular esta distancia los individuos se ordenan de forma creciente en cada uno de sus objetivos y se mide la distancia entre ellos. Ésta se obtiene calculando la distancia que tiene el individuo con sus dos vecinos más cercanos. Si el individuo se encuentra en un extremo del espacio objetivo, entonces se le asigna una distancia infinita, ya que a su lado no se encuentran más individuos. Después de calcular el nivel de no dominancia y la distancia de amontonamiento, se comienza con la selección de individuos que formarán parte de la población en la siguiente generación. Como se mencionó antes, NSGA-II implementa el elitismo por lo que generación a generación solo sobreviven los individuos con mejor aptitud, es decir, los de menor nivel de no dominancia. La diversidad en la nueva población depende de la distancia de amontonamiento. Cuando se está realizando la selección de individuos que sobreviven, si dos o más individuos presentan un mismo nivel de no dominación, entonces se selecciona a los que se encuentre en una zona menos poblada, es decir, al que tenga una distancia de amontonamiento mayor.

4.1 Algoritmo genético multiobjetivo propuesto

Hasta ahora hemos definido una función para cada uno de los objetivos que buscamos optimizar: tiempo, costo, estabilidad, desaprovechamiento y valor de liberación. De igual manera la función que calcula la aptitud de los individuos, la cual está basada en la no dominancia de soluciones. A continuación definiremos las partes del AG: representación de la solución como cromosoma, creación de la población y los operadores genéticos.

Cromosoma. Un cromosoma es una representación de la forma $a_1 a_2 a_3 \dots a_n$, en donde a cada posición i se le llama gen y a_i se le conoce como alelo. Cada gen

representa una HU y su valor es igual al sprint al que está asignada en la replaneación. El primer gen representa a la h_1, el segundo a la h_2 y así sucesivamente.

Creación de la población. Para la creación de la población, la primera vez que se ejecuta nuestro algoritmo, se crean los individuos a partir de la planeación inicial. Esto quiere decir que de la calendarización inicial se construye un cromosoma. Los demás miembros de la población son variaciones de este, los cuales se obtienen de mutar los valores al azar de la calendarización original.

Selección. Se realiza con un torneo aleatorio binario: se seleccionan dos individuos de la población al azar y gana el que tenga mejor aptitud. Si empatan en rango de no dominación, entonces se elige al que tenga una mayor distancia de amontonamiento.

Cruzamiento. El cruzamiento será en un solo punto y tendrá una probabilidad de 0.9. Se seleccionó este valor ya que presentó los mejores resultados preliminares, además que los autores de NSGA-II lo recomiendan. Cabe destacar que también se probó la cruza en dos puntos, pero los resultados no fueron satisfactorios.

Mutación. Para implementar este operador cada gen del cromosoma tendrá una probabilidad de ser mutado de 0.2. Se implementa una mutación gen a gen. Donde se recorren todo el cromosoma y en cada gen se obtiene una probabilidad entre 0 y 1 al azar. Si es menor o igual que 0.2, entonces se le asigna un nuevo sprint, distinto al que estaba asignado, el cual también obtenemos al azar. Se eligió este valor porque de igual manera presentó los mejores resultados en las pruebas preliminares.

Por la naturaleza probabilística de los AG, al aplicar alguno de los operadores genéticos a un individuo puede que este ya no sea factible. En nuestro algoritmo buscamos que todos los individuos de la población lo sean, por lo que implementamos un operador de reparación.

Reparación de cromosoma. En algunas ocasiones las soluciones dejan uno o más sprints vacíos. Por lo que, a pesar de que es un cromosoma válido, en los proyectos reales es algo indeseable. En esta etapa de reparación se valida al cromosoma y si presenta un sprint vacío, este se elimina. El procedimiento es el siguiente: Se identifica el sprint vacío. Se recorren las HU del sprint siguiente, al sprint vacío. Si hay más sprints vacíos se siguen recorriendo las HU para no dejar espacios. Si ya no hay sprints vacíos, se elimina el o los sprint vacíos que se recorrieron al final de la replaneación como resultado de la reparación.

5. Resultados

Al no encontrar trabajos que estudien la replaneación de proyectos enfocados en las metodologías ágiles, no se cuenta con casos de prueba públicos que permitan comparar nuestra propuesta. Por lo tanto, se crearon casos de prueba artificiales y los experimentos se dividieron en dos clases. La primera, con casos pequeños, 12 y 17 HU, para probar si el algoritmo ofrece soluciones coherentes. La segunda clase considera casos de prueba más grandes, desde 40 hasta 100 HU, para saber si el operador de reparación realmente aporta mejora al desempeño del algoritmo.

5.1. Experimentos con casos de prueba pequeños

Se crearon dos casos de prueba pequeños. El primero, 12 HU y 4 empleados, creado especialmente para mostrar el impacto de los objetivos. Los resultados para este caso de prueba fueron coherentes, al ser pequeños, visualmente pudimos observar el impacto de los objetivos y que el comportamiento fue el esperado al simular alguno de los eventos disruptivos. Los resultados fueron similares al segundo caso de prueba. El cual está formado de 17 HU y 4 empleados. Se muestra a continuación en la Fig. 2 el resultado después de simular un evento disruptivo que agrega una HU (HU 18).

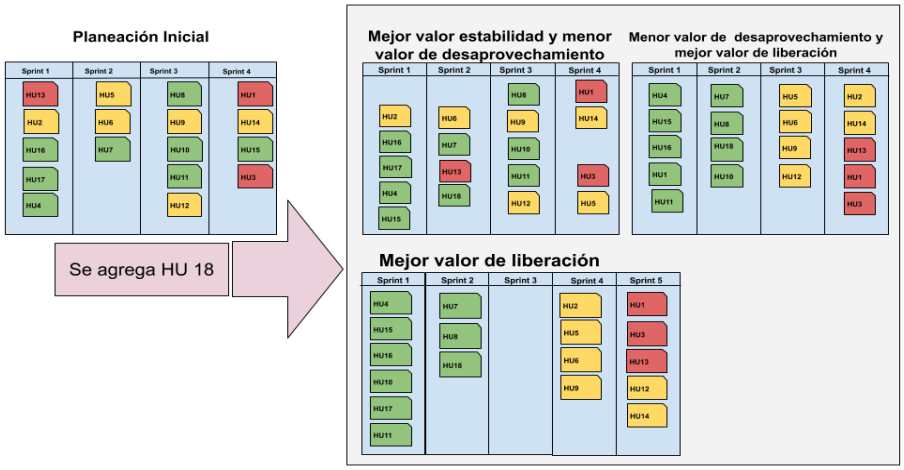

Fig. 2. A la izquierda se encuentra la planeación inicial, a la derecha tres resultados de replaneación: mejor valor de liberación, mejor estabilidad y mejor desaprovechamiento.

Como podemos observar en la parte inferior de la Fig. 2, la replaneación que presentó el máximo valor de liberación tiene un sprint vacío. La reparación, como antes se mencionó, ajusta las historias de usuario para que no queden sprint vacíos dentro de la replaneación. Al implementarla obtuvimos el resultado que se muestra en la Fig 3.

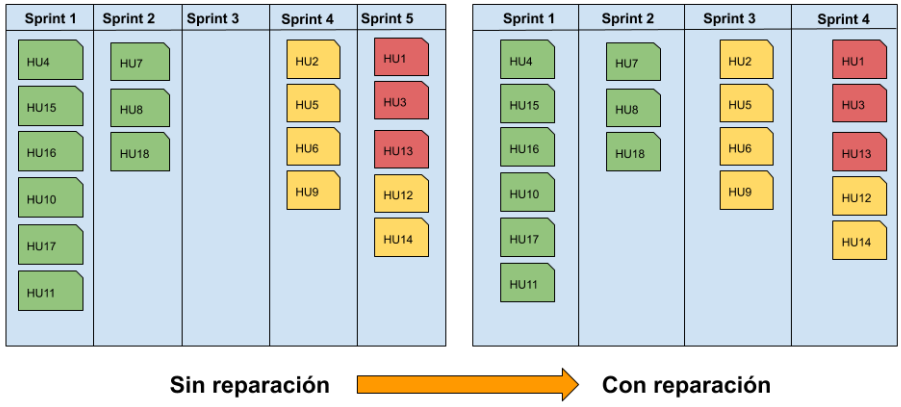

Fig. 3. Replaneación de liberación después de implementar la reparación de sprint vacío.

El algoritmo da como resultado un conjunto de propuestas de replaneación, las cuales presentan un equilibrio en sus objetivos o presentan el mejor valor encontrado para alguno de ellos. En la Fig. 2 se presenta, en la parte superior, dos ejemplos de equilibrio entre dos objetivos y en la parte inferior se muestra la planeación con el mejor valor de liberación. El valor de costo y tiempo son directamente proporcionales con el desaprovechamiento. Cuando el desaprovechamiento aumenta el tiempo también, ya que al desperdiciar mucho tiempo de desarrollo, el proyecto tardará más en desarrollarse. En el caso del costo es lo mismo, ya que al tardar más deberá gastarse en más sprints para terminar el proyecto.

5.2. Experimentos con casos de prueba grandes

Los casos de prueba para esta fase fueron creados con el generador de casos de prueba de Alba y Chicano [12], el cual se configuró para que creara aleatoriamente planeaciones con 40, 70 y 100 HU. Para el número de empleados, se consideraron 4, 5 y 6, respectivamente. Al ejecutar el generador de casos de prueba se obtuvieron planeaciones con hasta 16 sprints. En cada una de las pruebas se modelaron dos eventos disruptivos: se va un empleado y se agrega una nueva HU, ya que son los que tienen un mayor impacto negativo en la planeación.

Para comprobar si la reparación de sprint vacío realmente aporta al desempeño del algoritmo, Se obtiene un caso de prueba con un número de HU y de empleados con el generador. Después se simula un evento disruptivo y se ejecuta el algoritmo 10 veces. En cada ejecución tenemos la misma cantidad de HU, el mismo número de empleados, el mismo evento disruptivo y se aplica la reparación de sprint. De las 10 repeticiones se obtiene un promedio del tiempo de ejecución del algoritmo y los valores máximos de todas las ejecuciones para cada uno de los objetivos. Este procedimiento lo volvemos a realizar, pero ahora sin la reparación de sprint.

Ahora entonces tenemos 10 repeticiones con reparación y 10 sin ella, además de dos conjuntos de valores máximos. De los dos conjuntos volvemos a comparar y obtenemos los máximos globales de las 20 ejecuciones. Los máximos globales son los puntos de referencia con los cuales se calcula el hipervolumen de cada población resultado. Por último, calculamos el promedio del hipervolumen de las 10 poblaciones con reparación y el promedio de las 10 sin reparación. El resumen de los resultados de los experimentos se muestra en la Tabla 1. En la primera columna encontramos el número de HUs, la segunda muestra el número de empleados y en la tercera se muestra el evento disruptivo: **A** indica que se va un empleado y **B** representa que se agrega una nueva HU. Las últimas cuatro columnas representan, respectivamente, el promedio del tiempo de ejecución en segundos (t) y el promedio del hipervolumen para el algoritmo sin reparación y con reparación.

Los resultados muestran que la reparación de sprint vacío ayudó a que el algoritmo alcance un mayor hipervolumen. Podemos interpretar entonces que, cuando se aplica la reparación de sprint vacío, las replaneaciones de la población resultado presentan una mayor diversidad y mejores resultados en las funciones objetivo. Es decir, encuentran mejores replaneaciones, ya que, en general, la población converge a mejores resultados para los objetivos de tiempo, costo, estabilidad, desaprovechamiento y valor

de liberación. Podemos ver que, en la mayoría de los casos, el promedio del tiempo de ejecución aumenta al aplicarse la reparación, pero este no es considerable.

Tabla 1. Datos de los casos de prueba, el evento disruptivo y resumen de resultados obtenidos con el algoritmo genético multiobjetivo, sin y con operador de reparación.

No. HU	No. Emp.	Evento	Sin reparación		Con reparación	
			t	HV	t	HV
40	3	A	111.158	166716.55	109.904	184277.84
40	4	B	102.09	146908.69	104.49	177666.49
70	4	A	235.19	240939.09	243.59	301195.96
70	5	B	214.69	217033.16	212.79	254914.15
100	5	A	317.42	310596.38	339.34	315241.21
100	6	B	307.22	369386.78	313.48	463991.84

De hecho, para el primero y cuarto experimentos, el promedio del tiempo de ejecución con la reparación es menor que sin ella. Los resultados muestran que aplicar la reparación en el algoritmo da como resultado una población con un HV mayor, por lo tanto, el algoritmo tiene un mejor desempeño y no afecta significativamente el tiempo de ejecución.

6. Conclusiones

En este trabajo estudiamos la replaneación de liberaciones de proyectos ágiles de software como un problema de optimización multiobjetivo. La revisión bibliográfica indicó que no existen muchos trabajos sobre problemas de optimización en las metodologías ágiles de desarrollo de software. Aún más, no encontramos uno solo que desarrolle las características de éstas. Por lo que, como primera contribución, presentamos nuestra propuesta de un modelo para el problema de replaneación de liberaciones de proyectos ágiles de software. A diferencia de los trabajos encontrados en la revisión de la bibliografía, nuestro modelo presenta características específicas del desarrollo ágil como son: los conceptos de puntos de historia y velocidad de sprint.

Con este trabajo introducimos los objetivos de desaprovechamiento y valor de liberación. Además de implementar el tiempo extra de desarrollo de un equipo como un parámetro configurable, ya que cada equipo es distinto.

El problema de replaneación de liberaciones podemos considerarlo como una generalización del problema de asignación, el cual es un problema de la clase NP-difícil. Por lo tanto, para resolverlo, se propone un algoritmo genético multiobjetivo que implementa NSGA-II. Al ser un algoritmo basado en población, cubre la necesidad de obtener de manera rápida propuestas de replaneación de liberaciones después de ocurrir un evento disruptivo. En los resultados podemos ver que nuestro algoritmo presenta un conjunto de propuestas de replaneación de liberaciones en menos de 7 minutos para los casos más difíciles que fueron probados. El análisis de los resultados muestra que al aplicar la reparación en todos los casos el HV hipervolumen aumenta y por lo tanto el algoritmo tiene un mejor desempeño sin afectar apenas el tiempo de

ejecución. Estas soluciones presentan 100 HU, que abarcan 15 o 16 sprints, lo que podemos comparar con aproximadamente 240 días de desarrollo de un proyecto. En proyectos reales un experto generalmente presenta una propuesta de replaneación en mínimo 180 minutos [7], sin garantizar que es una buena elección.

Como trabajo futuro se propone obtener datos de proyectos reales y con esos datos ejecutar el algoritmo, simular los eventos disruptivos y verificar que las soluciones sean factibles en proyectos reales. De igual manera se buscará implementar otro tipo de heurísticas, como MOEA/D y SMS-EMOA. Se buscará implementar una segunda heurística para mejora de la población resultado (optimización local). Se investigarán otros indicadores de calidad que sean importantes para nuestros resultados y serán implementados. Por último, se buscará generar una herramienta de apoyo para los proyectos reales.

Referencias

1. Schwaber, K., Sutherland, J.: The scrum guide. The definitive guide to scrum: The rules of the game., vol. 268, Scrum org (2013)
2. Cohn, M.: Agile estimating and planning. Pearson Education (2005)
3. Jahr, M.: A hybrid approach to quantitative software project scheduling within agile frameworks. Project Management Journal 45(3), 35–45 (2014)
4. Roque, L., Araújo, A.A., Dantas, A., Saraiva, R., Souza, J.: Human Resource Allocation in Agile Software Projects Based on Task Similarities. In: Sarro F., Deb K. (eds) Search Based Software Engineering. Lecture Notes in Computer Science, vol. 9962, pp. 292–297. Springer, Cham (2016)
5. Nigar, N.: Model-based dynamic software project scheduling. In: Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering, pp. 1042–1045. ACM (2017)
6. Ge, Y.: Software project rescheduling with genetic algorithms. In: Artificial Intelligence and Computational Intelligence, 2009 International Conference, vol. 1, pp. 439–443. IEEE, Shanghai (2009)
7. Ge, Y., Xu, B.: Dynamic Staffing and Rescheduling in Software Project Management: A Hybrid Approach. PloS one 11(6), e0157104 (2016)
8. Shen, X., Minku, L.L., Bahsoon, R., Yao, X.: Dynamic software project scheduling through a proactive-rescheduling method. Transactions on Software Engineering 42(7), 658–686 (2016)
9. Shen, X.N., Minku, L.L., Marturi, N., Guo, Y.N., Han, Y.: A Q-learning-based memetic algorithm for multi-objective dynamic software project scheduling. Information Sciences, vol. 428, pp. 1–29 (2018)
10. Xiao, J., Osterweil, L.J., Wang, Q., Li, M.: Dynamic Resource Scheduling in Disruption-Prone Software Development Environments. In: Rosenblum D.S., Taentzer G. (eds) Fundamental Approaches to Software Engineering. Lecture Notes in Computer Science, vol 6013, pp. 107–122. Springer, Berlin, Heidelberg (2010)
11. Chen, W.N., Zhang, J.: Ant colony optimization for software project scheduling and staffing with an event-based scheduler. Transactions on Software Engineering 39(1), 1–17 (2013)
12. Alba, E., Chicano, J.F.: Software project management with GAs. Information Sciences, vol. 177, pp. 2380–2401 (2007)
13. Beume, N., Fonseca, C.M., López-Ibáñez, M., Paquete, L., Vahrenhold, J.: On the complexity of computing the hypervolume indicator. Transactions on Evolutionary Computation 13(5), 1075–1082 (2009)

14. Deb, K., Pratap, A., Agarwal, S., Meyarivan, T.A.M.T.: A fast and elitist multiobjective genetic algorithm: NSGA-II. Transactions on Evolutionary Computation 6(2), 182–197 (2002)