UNIVERSIDAD DE LOS ANDES
FACULTAD DE INGENIERÍA
DIVISIÓN DE ESTUDIOS DE POSTGRADO
MAESTRÍA EN COMPUTACIÓN

IMPLEMENTACIÓN DE METAHEURÍSTICAS PARA DAR SOLUCIÓN AL PROBLEMA DE PLANIFICACIÓN DE PROYECTOS DE SOFTWARE

Autora: Dahyana Carolina Nimo Parra
Tutor: Jhon Edgar Amaya Salazar

Trabajo de grado presentado ante la ilustre Universidad de Los Andes como requisito parcial para optar al grado de Magister Scientiae en Computación

Mérida, Septiembre 2012

UNIVERSIDAD DE LOS ANDES
FACULTAD DE INGENIERÍA
DIVISIÓN DE ESTUDIOS DE POSTGRADO
MAESTRÍA EN COMPUTACIÓN

IMPLEMENTACIÓN DE METAHEURÍSTICAS PARA DAR SOLUCIÓN AL PROBLEMA DE PLANIFICACIÓN DE PROYECTOS DE SOFTWARE

Autora: Dahyana Carolina Nimo Parra
Tutor: Jhon Edgar Amaya Salazar

Trabajo de grado presentado ante la ilustre Universidad de Los Andes como requisito parcial para optar al grado de Magister Scientiae en Computación

Mérida, Septiembre 2012

Agradecimientos

A Dios Todopoderoso por iluminar cada paso que doy en la vida, por ser mi guía incansable y por permitirme cumplir una meta más. A mi Esposo, mi mejor amigo y compañero de vida, gracias por tu compresión y apoyo incondicional, gracias por ser la alegría de mis días. A mi Madre por ser siempre mi ejemplo a seguir, por demostrarme el verdadero significado de la perseverancia y la superación. A mi Padre por protegerme cada día de mi vida y a mi papá por su siempre alentadora palabra.

Al Prof. Jhon Amaya, por su amistad, apoyo, tutela y acompañamiento durante el desarrollo de esta investigación, y en todas las oportunidades en las que he tenido el inmenso honor de trabajar a su lado.

A la Universidad de Los Andes, especialmente, a la División de Estudios de Postgrado, Maestría en Computación por su apoyo y formación constante a lo largo de mis estudios de Maestría.

A la Universidad Nacional Experimental del Táchira, especialmente al Departamento de Ingeniería Informática y al Laboratorio de Computación de Alto Rendimiento por su constante apoyo para el logro de esta investigación.

A Caniatech C.A., por las asesorías prestadas y por su incondicional colaboración para la obtención de los datos necesarios para llevar a término este trabajo.

A todos mis familiares y amigos, Gracias!

A mi Esposo, A mi Madre

Resumen

La planificación de proyectos de software, es un problema en el que se intenta determinar la correcta asignación de tareas a desarrolladores, para cumplir con los objetivos propuestos en un lapso de tiempo determinado, sin sobrepasar el presupuesto del proyecto y garantizando la calidad del producto a desarrollar. El objetivo de esta investigación es ofrecer un análisis de metaheurísticas para permitir ofrecer una solución a este problema, el cual fue representado como un problema de optimización combinatoria, y a partir de allí se desarrollaron diversas metaheurísticas como lo son el recocido simulado (SA), la búsqueda con vecindad variable (VNS) y los algoritmos genéticos (GA). Las pruebas de dichas metaheurísticas se realizaron sobre un conjunto de 18 casos de proyectos de software para cada enfoque de la función objetivo: reducción de costos ó reducción de tiempo asignando o no desarrolladores expertos. A partir de los resultados obtenidos se formuló un algoritmo híbrido basado en el VNS y el GA. Los resultados obtenidos por el algoritmo memético fueron superiores a los arrojados por el GA, sin embargo dichos resultados son equiparables a los obtenidos a través del VNS.

Palabras clave: *Planificación de proyectos de software, Metaheurísticas, Algoritmos Meméticos.*

Índice

Índice de Tablas ..... **viii**
Índice de Figuras ..... **xiv**

CAPÍTULO 1. INTRODUCCIÓN **1**

1.1 Introducción al Problema 1
1.2 Antecedentes 3
1.3 Justificación 5
1.4 Alcance 8
1.5 Objetivos 9
Objetivo General 9
Objetivos Específicos 9
1.6 Resultados Esperados 9
1.7 Organización de la Tesis 10

CAPÍTULO 2. PLANIFICACIÓN DE PROYECTOS **13**

2.1 Problemas de Optimización Combinatoria 13
Problemas de Planificación 14
2.2 Planificación de Proyectos de Software 15
Estimación de esfuerzo en proyectos de software 18

CAPÍTULO 3. FUNDAMENTOS DE METAHEURÍSTICAS **22**

3.1 Definición General 22
3.2 Clasificación de las metaheurísticas 25
Basadas en poblaciones o en métodos de trayectoria 25
De acuerdo con el uso de memoria 26
De acuerdo con la estructura de vecindad 26
3.3 Recocido Simulado 27
3.4 Búsqueda con vecindad variable 29
3.5 Algoritmos Genéticos 32

3.6 Algoritmos Meméticos 37

CAPÍTULO 4. DESARROLLO 40

4.1 Formulación del Problema 40
4.2 Modelado del Problema 44
4.3 Optimización del Problema 47
4.4 Implementación de la solución 51
4.5 Implementación del SA 58
4.6 Implementación del VNS 59
4.7 Implementación del GA 60
4.8 Interfaz Gráfica de Entrada y Salida 61

CAPÍTULO 5. RESULTADOS 70

5.1 Definición de casos de prueba 70
5.2 Evaluación de los resultados 73
Análisis de Sensibilidad del Algoritmo Genético 82
5.3 Implementación de un Algoritmo Memético 86
Análisis de sensibilidad del algoritmo memético 98
5.4 Comparación General 100

CAPÍTULO 6. CONCLUSIONES 107

CAPÍTULO 7. TRABAJO A FUTURO 109

REFERENCIAS 110

ANEXOS 115

ANEXO A. Recorrido Topológico de un Grafo 116
ANEXO B. Grafos de precedencia de tareas de los casos de prueba 117
ANEXO C. Desviaciones estándar de las pruebas realizadas 120

Índice de Tablas

Tabla 1. Estimación de proyectos de software con base en puntos de complejidad (PC) 21

Tabla 2. Lista de tareas T de un proyecto de software 52

Tabla 3. Lista de habilidades H requeridas para hacer un proyecto de software 52

Tabla 4. Lista de integrantes de un equipo de desarrollo E para proyectos de software 53

Tabla 5. Desarrolladores candidatos para cada tarea del proyecto representado por la matriz Xij 54

Tabla 6. Solución S1 – Planificación de proyecto de software 55

Tabla 7. Solución S2 - Planificación de proyecto de software 55

Tabla 8. Lista de tareas T de un proyecto de software – Grados por habilidad 57

Tabla 9. Lista de desarrolladores E– Grados por habilidad 57

Tabla 10. Tipos de Proyectos de Software 71

Tabla 11. Definición de Casos de Prueba 72

Tabla 12. Identificación de las Metaheurísticas 72

Tabla 13. Medias de los resultados obtenidos para la reducción de costos a partir de metaheurísticas simples 74

Tabla 14. Medias de los resultados obtenidos para la reducción de tiempo a partir de metaheurísticas simples 75

Tabla 15. Medias de los resultados obtenidos para la reducción de costos a partir de metaheurísticas simples asignando desarrolladores expertos 76
Tabla 16. Resultados de la prueba de Friedman para reducción de costos aplicando metaheurísticas simples..... 78
Tabla 17. Resultados de la prueba de Friedman para reducción de tiempo aplicando metaheurísticas simples..... 78
Tabla 18. Resultados de la prueba de Friedman para reducción de tiempo aplicando metaheurísticas simples asignando desarrolladores expertos 78
Tabla 19. Resultados de la prueba de Holm para reducción de costos en proyectos pequeños aplicando metaheurísticas simples. Algoritmo de Control: VNS 79
Tabla 20. Resultados de la prueba de Holm para reducción de costos en proyectos medianos aplicando metaheurísticas simples. Algoritmo de Control: VNS..... 79
Tabla 21. Resultados de la prueba de Holm para reducción de costos en proyectos grandes aplicando metaheurísticas simples. Algoritmo de Control: VNS..... 80
Tabla 22. Resultados de la prueba de Holm para reducción de tiempo en proyectos pequeños aplicando metaheurísticas simples. Algoritmo de Control: VNS 80
Tabla 23. Resultados de la prueba de Holm para reducción de tiempo en proyectos medianos aplicando metaheurísticas simples. Algoritmo de Control: VNS..... 80
Tabla 24. Resultados de la prueba de Holm para reducción de tiempo en proyectos grandes aplicando metaheurísticas simples. Algoritmo de Control: VNS..... 81

Tabla 25. Resultados de la prueba de Holm para reducción de tiempo asignando desarrolladores Expertos en proyectos pequeños aplicando metaheurísticas simples - Algoritmo de Control: VNS 81
Tabla 26. Resultados de la prueba de Holm para reducción de tiempo asignando desarrolladores expertos en proyectos medianos aplicando metaheurísticas simples - Algoritmo de Control: VNS 81
Tabla 27. Resultados de la prueba de Holm para reducción de tiempo asignando desarrolladores expertos en proyectos grandes aplicando metaheurísticas simples - Algoritmo de Control: VNS 82
Tabla 28. Identificación de las versiones del Algoritmo Genético para el análisis de sensibilidad (Reducción de Tiempo) 83
Tabla 29. Medias de los resultados obtenidos para reducción de tiempo con distintas versiones del algoritmo genético (Análisis de Sensibilidad) 84
Tabla 30. Resultados de la prueba de Holm para reducción de tiempo con distintas versiones del algoritmo genético - Algoritmo de Control: GA_I 85
Tabla 31. Identificación del Algoritmo Memético 87
Tabla 32. Medias de los resultados obtenidos para la reducción de costos a partir de la aplicación del algoritmo memético 88
Tabla 33. Medias de los resultados obtenidos para la reducción de tiempo a partir de la aplicación del algoritmo memético 89
Tabla 34. Medias de los resultados obtenidos para la reducción de tiempo asignando desarrolladores expertos a partir de la aplicación del algoritmo memético 90

Tabla 35. Resultados de la prueba de Friedman para reducción de costos aplicando el algoritmo memético 91
Tabla 36. Resultados de la prueba de Friedman para reducción de tiempo aplicando el algoritmo memético 91
Tabla 37. Resultados de la prueba de Friedman para reducción de tiempo asignando desarrolladores expertos aplicando el algoritmo memético 91
Tabla 38. Resultados de la prueba de Holm para reducción de costos en proyectos pequeños aplicando el algoritmo memético. Algoritmo de Control: VNS 92
Tabla 39. Resultados de la prueba de Holm para reducción de costos en proyectos medianos aplicando el algoritmo memético. Algoritmo de Control: VNS 92
Tabla 40. Resultados de la prueba de Holm para reducción de costos en proyectos grandes aplicando el algoritmo memético. Algoritmo de Control: VNS 93
Tabla 41. Resultados de la prueba de Holm para reducción de tiempo en proyectos pequeños aplicando el algoritmo memético. Algoritmo de Control: VNS 93
Tabla 42. Resultados de la prueba de Holm para reducción de tiempo en proyectos medianos aplicando el algoritmo memético. Algoritmo de Control: VNS 94
Tabla 43. Resultados de la prueba de Holm para reducción de tiempo en proyectos grandes aplicando el algoritmo memético. Algoritmo de Control: VNS 94
Tabla 44. Resultados de la prueba de Holm para reducción de tiempo en proyectos pequeños con asignación de desarrolladores expertos aplicando el algoritmo memético. Algoritmo de Control: VNS 95

Tabla 45. Resultados de la prueba de Holm para reducción de tiempo en proyectos medianos con asignación de desarrolladores expertos aplicando el algoritmo memético. Algoritmo de Control: VNS 95
Tabla 46. Resultados de la prueba de Holm para reducción de tiempo en proyectos grandes con asignación de desarrolladores expertos aplicando el algoritmo memético. Algoritmo de Control: VNS 96
Tabla 47. Diagramas de Cajas y Bigotes de los resultados del algoritmo memético para reducción de costos y tiempo 97
Tabla 48. Identificación de las versiones del Algoritmo Memético para el..... 98
Tabla 49. Medias de los resultados obtenidos para reducción de tiempo con distintas versiones del algoritmo memético (Análisis de Sensibilidad) 99
Tabla 50. Comparación de los resultados obtenidos para reducción de tiempo en la comparación de metaheurísticas y el algoritmo memético 102
Tabla 51. Resultados de la prueba de Friedman para reducción de tiempo en la comparación de metaheurísticas y el algoritmo memético 102
Tabla 52. Resultados de la prueba de Holm para reducción de tiempo (Proyectos pequeños) en la comparación de metaheurísticas y el algoritmo memético. Algoritmo de Control: VNS 103
Tabla 53. Resultados de la prueba de Holm para reducción de tiempo (Proyectos medianos) en la comparación de metaheurísticas y el algoritmo memético. Algoritmo de Control: VNS 103

Tabla 54. Resultados de la prueba de Holm para reducción de tiempo (Proyectos grandes) en la comparación de metaheurísticas y el algoritmo memético. Algoritmo de Control: VNS 103
Tabla 55. Diagramas de Cajas y Bigotes de la comparación de metaheurísticas y el algoritmo memético para reducción de tiempo 104
Tabla 56. Diagramas de Cajas y Bigotes de comparación de las estrategias implementadas para todos los casos de prueba 106

Índice de Figuras

Figura 1. Triángulo del Proyecto 16

Figura 2. Ciclo de Vida del Proyecto 17

Figura 3. Clasificación de Técnicas de Optimización. Fuente: Chicano (2007) 23

Figura 4. Pseudocódigo del SA 29

Figura 5. Pseudocódigo del VNS 31

Figura 6. Pseudocódigo de un Algoritmo Genético 37

Figura 7. Estructura del equipo de desarrollo 43

Figura 8. Estructura de las tareas de un proyecto de software 43

Figura 9. Grafo de Precedencia de tareas *A* de un proyecto de software 53

Figura 10. Interfaz de entrada 62

Figura 11. Estructura del archivo de entrada de Habilidades 62

Figura 12. Estructura del archivo de entrada de Desarrolladores 62

Figura 13. Estructura del archivo de entrada de Tareas 63

Figura 14. Interfaz de Salida – Descripción del Problema 65

Figura 15. Interfaz de Salida – Descripción de la Solución 65

Figura 16. Interfaz de Salida – Descripción de la Planificación 66

Figura 17. Interfaz de Salida – Vista Parcial del Grafo de precedencia de tareas (TPG) 67

Figura 18. Interfaz de Salida – Carga de trabajo por desarrollador 68

Figura 19. Interfaz de Salida – Costos por desarrollador 68

Figura 20. Interfaz de Salida – Planificación en función del tiempo 69

Figura 21. Diagrama de Cajas y Bigotes: Versiones del AG en el análisis de sensibilidad 85

Figura 22. Pseudocódigo del Algoritmo Memético 87

Figura 23. Diagrama de Cajas y Bigotes: Versiones del AM en el análisis de sensibilidad 100

Capítulo 1. Introducción

1.1 Introducción al Problema

La constante búsqueda de soluciones a problemas de la vida cotidiana a través de herramientas computacionales, es una tendencia que a lo largo del tiempo se ha convertido en el estándar de facto para prácticamente cualquier área de mercado. En tal sentido, ha sido necesario mejorar las estrategias existentes y diseñar nuevas estrategias computacionales, en busca de ofrecer las capacidades y opciones necesarias para dar soporte a problemas de mayor complejidad, que puede estar dada por el problema en sí mismo o por la selección de una solución entre el universo de posibles soluciones del problema en cuestión (Martí, 2003), los cuales son conocidos como problemas de optimización combinatoria.

La complejidad de los problemas de optimización combinatoria (COP) viene dada por la forma de explorar el universo de posibles soluciones para un problema en particular dado el amplio número de elementos que lo componen (Belén, Moreno, & Moreno, 2003). Tal universo puede llegar a ser demasiado amplio en un espacio discreto, por lo que la exploración para seleccionar una solución en particular que cumpla con las condiciones esperadas, es un proceso que requiere del uso de las estrategias apropiadas (Martí, 2003). Por su parte, las metaheurísticas son entendidas como la combinación de diferentes métodos heurísticos a un nivel más alto para

conseguir una exploración del espacio de búsqueda eficiente y efectiva (Glover, 1986), requieren la definición de una serie de parámetros y de un conjunto de pasos según los cuales es posible recorrer ampliamente un espacio de búsqueda sin tener que evaluar todos y cada uno de sus elementos, por lo que son ampliamente utilizadas para buscar soluciones a este tipo de problemas (Belén, Moreno, & Moreno, 2003). Estas estrategias, a diferencia de los métodos exactos no garantizan la obtención del óptimo global, sin embargo, son considerados métodos genéricos que ofrecen soluciones de buena calidad en un tiempo moderado (Chicano, 2007).

Por su parte, la planificación de proyectos de software es un problema en el que se intenta determinar la correcta asignación de desarrolladores a tareas de acuerdo con las capacidades requeridas, para cumplir con los objetivos propuestos en un lapso de tiempo determinado, por lo que puede ser representado como un problema de COP ---específicamente de planificación, pues para un mismo proyecto puede existir una gran cantidad de planificaciones posibles y factibles, cada una con un costo y duración asociadas. En investigaciones como la propuesta por Chicano (2007), se emplean diversas metaheurísticas para abordar la planificación de proyectos de software, lo cual constituye un importante punto de referencia para este proyecto, cuyo principal objetivo es implementar diversas metaheurísticas como: los algoritmos genéticos (Larrañaga, Abdelmalik, & Iñaki, 2010), la búsqueda de vecindad variable (Sarasola, Doerner, & Alba, 2012) y el recocido simulado (Dowsland & Adenso, 2003), para encontrar en un lapso de tiempo razonable

planificaciones de proyectos de software que busquen emplear los recursos disponibles para disminuir el tiempo o el costo de ejecución del proyecto según sea el caso, por lo que se enmarca en un área de interés actual, pues aún cuando se han determinado diversas metodologías y técnicas para mejorar dicho proceso (Biolchini, Gomes, Cruz, & Horta, 2005), los resultados obtenidos todavía no concuerdan del todo con los esperados (Mian, Conte, Natali, Biolchini, & Travassos, 2005).

En tal sentido, esta investigación atiende el desarrollo de algoritmos que buscan ser más eficientes para resolver problemas complejos, como lo es la planificación de proyectos de software, formulado como un problema de optimización combinatoria, considerando además los resultados arrojados por diversas investigaciones (Martí, 2003) en los que se demuestra que la combinación de diversas metaheurísticas permite la creación de estrategias más robustas en cuanto a rendimiento y calidad de las soluciones obtenidas, por lo que también se incluye el diseño de un mecanismo de hibridación o algoritmo memético, que permita implementar y posteriormente evaluar este tipo de estrategias.

1.2 Antecedentes

En el trabajo realizado por Cervantes (2010) se aborda el problema de planificación específicamente para escenarios con recursos limitados, buscando proponer, diseñar y desarrollar nuevos métodos metaheurísticos para obtener una asignación optimizada de recursos para dicho caso. Con tal fin se contemplan dos

versiones del problema de planificación: la versión estándar, también conocida como versión de modo único (RCPSP), en la que se busca minimizar la duración del proyecto cuando las actividades que lo componen no pueden interrumpir su ejecución y están sujetas exclusivamente a relaciones de precedencia de tipo fin-inicio, es decir tienen un único modo de ejecución, con una duración determinada y un consumo dado de recursos; y la versión multi-modo (MRCPSP), en la que cada una de las actividades puede presentar distintas posibilidades o modos diferentes de ejecución y además puede contar con recursos no renovables y doblemente limitados. Para ambas versiones del problema se implementaron un conjunto de metaheurísticas, con un enfoque basado en los algoritmos genéticos, debido a que los resultados que se han reportado en la literatura los muestran como promisorios para hallar soluciones de alta calidad aún en problemas de grandes dimensiones. Asimismo, la implementación híbrida de los métodos desarrollados ofreció mejores resultados para los escenarios de prueba de mayores dimensiones.

En el mismo orden de ideas, existen trabajos en lo que se refiere a implementación de metaheurísticas para ofrecer soluciones a la planificación de proyectos de software. Tal es el caso del estudio realizado por Chicano (2007), correspondientes a metaheurísticas e ingeniería del software. En este estudio se abordan tres problemas de la ingeniería del software: planificación de proyectos de software, generación automática de casos de prueba y búsqueda de violaciones de propiedades de seguridad en sistemas concurrentes, para cada uno de los cuales se

evalúa la implementación de diversas metaheurísticas de acuerdo con las particularidades de cada caso (Chicano, 2007). Específicamente, para la planificación de proyectos de software fundamentada en el *Project Scheduling Problem* ---PSP, se implementó un algoritmo genético a partir del cual se generan planificaciones que buscan reducir la duración del proyecto. Los aportes obtenidos a partir de esta implementación permiten determinar la factibilidad de la utilización de esta técnica para la resolución de este tipo de problemas. A partir de las pruebas realizadas con base en un generador de instancias diseñado por el autor, se determinaron importantes conclusiones acerca de las posibles relaciones existentes entre las variables consideradas en las instancias del problema. Sin embargo, resulta importante destacar que el modelo utilizado en este trabajo se refiere al exacto propuesto a través del PSP, por lo que claramente el gerente de proyectos posee diversas limitaciones para gestionar proyectos de la vida real mediante el mismo.

1.3 Justificación

La planificación de proyectos de software es una de las actividades que actualmente representa mayores problemas y a su vez mayor importancia para los gerentes de proyectos de esta área (Abdel & Madnick, 2001), pues la programación de las tareas que se deben cumplir en función del tiempo y con los recursos disponibles, es una labor crítica para el éxito de cualquier proyecto de software. Ciertamente, la realización de proyectos de cualquier tipo involucra una fase previa

de planificación que resulta vital para el cumplimiento de los objetivos (Beltran, et al., 2005). Sin embargo, en lo que a planificación de proyectos de software se refiere, vale la pena preguntarse: por qué el común denominador para dichas planificaciones es culminar los proyectos después de la fecha pautada originalmente (Abdel & Madnick, 2001).

El desarrollo de software es un proceso que difiere de otros procesos de producción por diversas razones. Por ejemplo, solventar un error de una tarea determinada, realizada originalmente por un desarrollador "A", le llevaría 7 veces a un desarrollador "B" resolverlo, con respecto a lo que le llevaría al mismo desarrollador "A" (Grompone, 1996). En este sentido, la reasignación de desarrolladores para resolver tareas que no han sido tratadas por ellos puede tener efectos en la duración, y por ende en el costo total del proyecto. De la misma manera, otros factores como: la magnitud del proyecto, el conocimiento de las tecnologías requeridas, el tipo de tareas y el nivel de información que se tiene para realizarlas, los cambios introducidos o solicitados por el cliente durante su desarrollo, la experiencia del desarrollador, entre otros, son algunos de los puntos críticos de la planificación de proyectos de software, que la hacen un proceso complejo que además puede variar de un proyecto a otro (Abdel & Madnick, 2001).

Dicho en otras palabras, la planificación de proyectos de software consiste en definir qué desarrolladores se asignan a cada tarea y cuándo ésta debería llevarse a cabo (Chicano, 2007), labor que actualmente realiza el gerente de proyectos de una

manera casi empírica (Junk, 2000), considerando para ello las habilidades y la disponibilidad de cada empleado, la información necesaria para realizar la tarea, posibles necesidades de recursos extras, etc. (Junk, 2000). Por tanto, el objetivo de dicha planificación se enfoca en (i) minimizar la duración del proyecto con miras a cumplir con las fechas de entrega pautadas o (ii) minimizar el costo de producción, (iii) garantizando de la mejor manera posible en cualquier de los casos la calidad del producto realizado (Chicano, 2007).

En consecuencia, la planificación idónea reconcilia estos tres objetivos, sin embargo, en la realidad suelen ser incluso excluyentes, pues conseguir una buena combinación de recursos y tareas en función del tiempo para proyectos de gran magnitud bajo dichas premisas, requiere de una inversión de tiempo y esfuerzo que por lo general no es factible para un proyecto de software promedio. En este sentido, este problema puede ser visto como un problema de optimización combinatoria, específicamente como un problema de planificación (Beltran, et al., 2005), buscando que la implementación de este tipo de herramientas, que mediante el empleo de un esfuerzo computacional razonable permita obtener una solución que aunque no sea la óptima, pueda satisfacer las necesidades del gerente de proyectos.

Es importante, destacar que el problema de planificación abordado puede ser considerado un caso particular de planificación dinámica (Pinedo, 1983), donde deben considerarse que existen sistemas cuya carga de trabajo no es fija, debido a diferentes elementos como por ejemplo, un entorno es cambiante, tareas se

crean/destruyen dinámicamente, la aparición de tareas en ráfagas o por la falla de una parte del sistema falla. Bajo esa perspectiva, la planificación debe realizar en tiempo de ejecución y en función de los atributos de las tareas, debido a que no se conoce a priori las necesidades de tareas (el número de tareas no es fijo y sus atributos son arbitrarios ---uso de recursos, tiempo de cómputo, plazos de respuesta, relaciones de precedencia, etc.---), por lo cual la complejidad del problema se hace mayor. Se ha abordado la planificación dinámica con diferentes enfoques algorítmicos, como por ejemplo evolución diferencial (Liu, 2011), aprendizaje reforzado (Vengerov, 2005), entre otros.

1.4 Alcance

Esta investigación abarca la implementación de un conjunto de metaheurísticas para dar una posible solución al problema de planificación de proyectos de software, formulado como un problema de optimización combinatoria a partir del PSP. Asimismo, contempla la implementación de un algoritmo híbrido con base en las metaheurísticas implementadas de acuerdo con los resultados obtenidos.

1.5 Objetivos

Objetivo General

Implementar metaheurísticas para dar solución al problema de planificación de proyectos de software.

Objetivos Específicos

- Analizar la documentación disponible acerca de la aplicación de metaheurísticas para dar solución a problemas de planificación.
- Modelar el problema de planificación de proyectos de software en función de las variables a optimizar.
- Seleccionar e implementar las metaheurísticas para dar solución al problema de planificación de proyectos de software.
- Realizar pruebas de funcionamiento y rendimiento sobre las metaheurísticas implementadas.
- Evaluar los resultados obtenidos a partir de las metaheurísticas codificadas.

1.6 Resultados Esperados

Se espera implementar un conjunto de metaheurísticas para dar solución al problema de la planificación de proyectos de software buscando minimizar el tiempo

o el costo de un proyecto determinado, así como también formular un algoritmo memético con base en los resultados obtenidos.

1.7 Organización de la Tesis

Capítulo 1. Introducción. Incluye la descripción general del problema y sus correspondientes antecedentes, justificación objetivos y alcance. En la descripción del problema se denotan los principales factores relacionados con el mismo y el porqué puede ser abordado como un problema de optimización combinatoria.

Capítulo 2. Planificación de proyectos. En este capítulo se presenta una breve descripción de los problemas de optimización combinatoria, enfocando principalmente los problemas de planificación. Posteriormente, se describen los principales aspectos relacionados con la planificación de proyectos de software y la estimación de esfuerzo en los mismos.

Capítulo 3. Fundamentos de Metaheurísticas. Se realiza una revisión de los aspectos teóricos de las metaheurísticas y su respectiva clasificación. Posteriormente, se presenta con detalle la descripción de las metaheurísticas implementadas en esta investigación.

Capítulo 4. Desarrollo. Contiene la descripción de cada una de las actividades llevadas a cabo durante la ejecución de esta investigación las cuales obedecen a las fases propuestas por (Talbi, 2009), correspondientes a una

metodología aplicada a la optimización de modelos orientados a metaheurísticas, la cual abarca desde la concepción inicial del problema, hasta su implementación y proceso de mejora, de la siguiente manera.

- **Formulación del problema:** En esta fase se contemplan las actividades correspondientes al conocimiento del problema, variables asociadas, factores internos y externos, así como también la definición de los objetivos que se buscan resolver a partir de su planteamiento. Para efectos de esta investigación, el problema se formuló a partir de las principales necesidades de los gerentes de proyectos de software referidas a minimización de costos o duración de dichos proyectos. Asimismo, se determinaron los posibles escenarios de prueba.

- **Modelado del problema**: El modelado del problema, referido a la construcción de un modelo matemático abstracto que describe la situación planteada, se realizó a partir del PSP, entendido como un modelo de descripción general de problemas de planificación. A partir del mismo, se realizaron las adaptaciones necesarias para ajustar el modelo a la planificación de proyectos de software específicamente.

- **Optimización del problema:** Corresponde a la fase en la que se cotejan los objetivos determinados en la formulación del problema con el modelo matemático generado. Se analiza la función objetivo en términos de reducción de costos, tiempo de desarrollo y participación de expertos en el grupo de

desarrolladores seleccionados, en contraparte con las restricciones del problema.

- **Implementación de la solución**: a partir de la optimización del problema se procede a la implementación de las metaheurísticas seleccionadas con base en el modelo matemático planteado. De esta manera se obtiene una primera versión de los resultados de cada una de ellas, cuya comparación permitirá la formulación de una estrategia hibrida para buscar solución al problema objeto de estudio.

Capítulo 5. Resultados. Se presentan y analizan los resultados obtenidos a partir de las metaheurísticas desarrolladas para todos los casos de prueba, así como también la formulación del algoritmo memético diseñado a partir de dichos resultados.

Capítulo 6. Conclusiones. En el que se presentan las conclusiones obtenidas a partir del análisis de resultados obtenidos con respecto a la implementación de metaheurísticas para dar solución al problema de planificación de proyectos de software.

Capítulo 7. Trabajo a Futuro. Finalmente en este capítulo se presentan las propuestas para futuras investigaciones a partir de este trabajo.

Capítulo 2. Planificación de Proyectos

En este capítulo, se describen los aspectos más importantes del problema de planificación de proyectos de software visto como un problema de optimización combinatoria, indicando algunas definiciones básicas, así como también algunas particularidades de la planificación de proyectos de software específicamente, como lo son las técnicas de estimación de esfuerzo de las fases de planeación.

2.1 Problemas de Optimización Combinatoria

Los problemas de optimización combinatoria se refieren al tipo de problemas cuyo objetivo es encontrar el máximo (o el mínimo) de una determinada función sobre un conjunto finito de soluciones, llamado *S*, el cual está generalmente constituido por un número de muy elevado de soluciones, por lo que evaluarlos a todos y cada uno de ellos para tratar de encontrar el mejor resulta en la mayoría de los casos impracticable (Chicano, 2007). Así por ejemplo, un tablero de ajedrez que consta de 64 casillas en las que se colocan las treinta y dos piezas del juego, un excursionista que quiere llenar su mochila de la forma más eficaz aún cuando deba dejar algunos elementos fuera, o la empresa que necesita ejecutar una serie de tareas de manera tal que puedan ser realizadas en el menor tiempo posible, son algunos ejemplos de problemas de optimización combinatoria, en los cuales se distinguen dos tipos de elementos comunes a todas las situaciones: (i) un conjunto de objetos, casos,

personas, etc., que se han de colocar en distintas posiciones y (ii) una familia de lugares en las que se deben colocar dichos objetos (García, 2000).

De esta manera, la colocación de objetos en sus lugares se denomina configuración, y es equivalente a una solución al problema abordado. El conjunto de todas las configuraciones posibles conforman el conjunto *S*. Intentar encontrar la mejor configuración corresponde a un problema de optimización combinatoria, el cual puede ser mono-objetivo cuando sobre el conjunto *S* se construye una sola función de valor (o función objetivo) y multiobjetivo cuando es necesario optimizar varios objetivos.

Problemas de Planificación

Los problemas de planificación, también conocidos como problemas de *Scheduling*, corresponden al área de problemas de optimización combinatoria, específicamente de satisfacción de restricciones en los que la búsqueda de soluciones debe optimizar determinados criterios de eficiencia (Cervantes, 2010). Por ejemplo, los problemas relacionados con la asignación de citas en el sector hospitalario, el establecimiento de horarios en una universidad o la planificación de proyectos, corresponden a instancias de problemas de planificación.

Específicamente, en el problema de planificación de proyectos se considera un conjunto de actividades relacionadas entre sí, un conjunto de recursos y un conjunto de medidas de desempeño, con miras a encontrar la mejor manera de asignar los

recursos a las actividades, siendo ésta la que maximice las medidas de desempeño establecidas. Existen diversas variantes de este problema en los que las actividades tienen asignada una determinada duración y un determinado consumo de recursos. Sin embargo, la versión de modo único (RCPSP) corresponde a aquellos proyectos en los que sus actividades tienen un único modo de ejecución, no pueden ser interrumpidas, y están sujetas las precedencias de tipo fin – inicio para poder ser ejecutadas, mientras que en la versión multi-modo (MRCPSP) las actividades de los proyectos presentan distintas formas de ejecución (Cervantes, 2010).

Este tipo de problemas pertenece a la clase de problemas NP-duros, por lo que se ha determinado que encontrar la solución óptima requiere tiempos muy altos para proyectos de 30 o más actividades (Blazewicz & Lenstra, 1983), incluso cuando hay un único modo de ejecución. La formulación del problema de planificación de proyectos se ha adaptado a diversos campos de aplicación, con base en las diferentes variantes que puede presentar un tipo de proyecto en particular de acuerdo con su respectivo proceso de producción, como los proyectos de manufactura, desarrollo de software, etc. (Chicano, 2007).

2.2 Planificación de Proyectos de Software

La planificación de proyectos de software es una instancia particular del problema de planificación de proyectos en la que se busca realizar las adaptaciones correspondientes a esta área de estudio. En primer lugar, se entiende por proyecto el

"*esfuerzo temporal realizado para crear un producto, servicio o resultado único*" (Duncan, 2004), dicho en otras palabras, esta definición corresponde al conjunto de actividades interdependientes que se deben desarrollar en un lapso de tiempo determinado para lograr los objetivos propuestos, buscando hacerlo en el menor *tiempo*, con el menor *costo* y la mayor *calidad* posible. Estos factores: tiempo, costo y calidad, conforman lo que se ha llamado el triángulo del proyecto (Ver Figura 1), cuyos vértices vienen a ser los pilares fundamentales de cualquier proyecto de software: que (i) debe ser finalizado parcial o completamente antes de una fecha límite, (ii) no debe sobrepasar un presupuesto determinado y (iii) debe poseer una calidad mínima aceptable en el desarrollo de cada uno de sus requerimientos. La conciliación de dichos factores en la realidad es una tarea correspondiente a la gestión de proyectos *(Project Management)*, es decir, el gerente de proyectos es la persona encargada de administrar los recursos de manera tal que se pueda culminar todo el trabajo requerido en el proyecto dentro del alcance, el tiempo y costo definidos, buscando asegurar además la calidad del producto a generar.

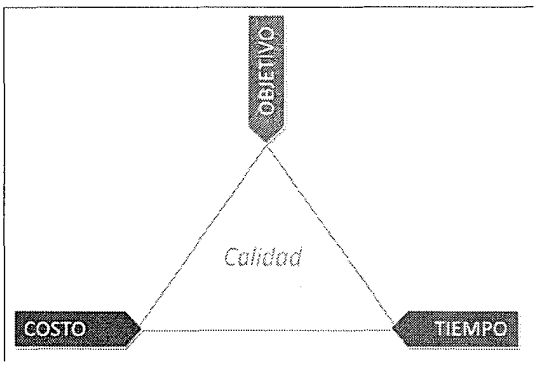


Figura 1. Triángulo del Proyecto

Asimismo, es bien sabido que existen múltiples metodologías para el desarrollo de proyectos de software (Biolchini, Gomes, Cruz, & Horta, 2005). Sin embargo, las fases establecidas por la gran mayoría de ellas involucran actividades correspondientes a la planeación, planificación, ejecución, control y terminación del proyecto, tal como se indica en la Figura 2. En la fase de planificación de proyectos específicamente, se realiza la asignación de recursos a tareas en función del tiempo disponible para realizarlo, actividad que tal como se ha indicado es responsabilidad del gerente de proyectos. Dicha actividad tiene una gran importancia para la realización exitosa del mismo, pues las decisiones tomadas en este punto pueden tener repercusiones directas en la duración, costo y calidad del proyecto tal como se ha indicado.

Figura 2. Ciclo de Vida del Proyecto

Esta investigación se enfoca en la fase de planificación de proyectos de software, teniendo en cuenta las restricciones particulares de esta área de estudios descritas más adelante. Sin embargo, el éxito de esta fase depende de las salidas generadas en lo que corresponde a la planeación, particularmente de las estimaciones realizadas para cada una de las tareas o actividades que se deben llevar a cabo durante

la ejecución del proyecto. En tal sentido, a continuación se describen algunos de los aspectos más importantes en cuanto a la estimación de esfuerzo para proyectos de software.

Estimación de esfuerzo en proyectos de software

El proceso de estimación de esfuerzo en proyectos de software está sujeto a un conjunto de dificultades (Kniberg, 2007): información imprecisa acerca del proyecto o de las capacidades de la organización que lo desarrollará, caos en el proceso de desarrollo del producto, cambios en el equipo de desarrollo, etc., por lo que el gerente de proyectos debe tratar de realizar dichas estimaciones con base en la información disponible, pero permitiendo un margen de error para los posibles cambios que ocurran. Anteriormente, dicha estimación se realizaba con base en métricas como la cantidad de líneas de código estimadas para realizar una funcionalidad determinada, lo que a su vez influía en la clasificación del tamaño de los proyectos de software. Sin embargo, dada las diferencias ofrecidas por los lenguajes de programación actuales, esta métrica se considera en desuso, pues si a la empresa A le puede llevar 8 mil líneas de código resolver un determinado problema en un lenguaje de programación, a la empresa B le puede llevar 5 mil líneas para resolver el mismo problema en otro lenguaje. Es importante destacar que bajo este escenario no es posible afirmar que la empresa B podrá terminar el proyecto antes que la empresa A, pues esto depende estrictamente de la productividad de sus desarrolladores.

[1]Por otra parte, en el proceso tradicional de estimación de esfuerzo en proyectos de software (Biolchini, Gomes, Cruz, & Horta, 2005), el gerente o líder del equipo de desarrollo indica el tiempo disponible para la realización de cada actividad, tomando en cuenta su propia experiencia y la posible experticia de la persona a la que asignará dicha labor, así como también la información disponible para llevarla a cabo. Sin embargo, en la actualidad otras técnicas correspondientes a las metodologías ágiles de desarrollo de software proponen la estimación grupal del esfuerzo que cada actividad requiere. La estimación con base en puntos de complejidad[1] (Durán, 2003) es un método que permite basarse en los requerimientos propios del usuario, y no en la tecnología que se va a utilizar, lo que la convierte en una métrica basada en la funcionalidad, independiente de la tecnología, simple y consistente, a partir de la cual es posible además determinar el tamaño del software tal como se indica en la Tabla 1, en donde se especifica un ejemplo de las estimaciones de una empresa particular en comparación con la industria mundial de software. La información se presenta en tres indicadores: productividad, esfuerzo y duración, así por ejemplo la productividad de los proyectos pequeños para la industria de software se estima en 10.73, lo cual indica la relación los puntos de complejidad asignados al proyecto y tiempo invertido en la conclusión del mismo. Se puede observar que los datos presentados no son lineales con respecto al comportamiento específico de una empresa determinada, lo cual

[1] Un punto de complejidad asociado a una tarea, es un número entero ($i \in \mathbb{N}^+$) que indica la complejidad de la misma con respecto a otras tareas de un mismo proyecto. Los puntos de complejidad son asignados por uno o varios integrantes del equipo de desarrollo dependiendo de su grado de experiencia y/o experticia.

indicaría que esta empresa ha realizado una peor estimación de los proyectos medianos y ha conllevado a mayor esfuerzo para la realización de los proyectos, debido a que la planificación depende del equipo de desarrollo involucrado y de las estimaciones realizadas por los mismos.

El *Poker Planning* (Kniberg, 2007) es un ejemplo de estas técnicas, según la cual el equipo de desarrollo puede determinar en conjunto el esfuerzo que requiere cada actividad, realizando diversos procesos de negociación para tal fin. En este caso, la estimación no se realiza directamente en proporción del tiempo a emplear sino en función de la complejidad de cada tarea expresada en puntos. Cada integrante del equipo de desarrollo puede expresar su estimación en cuanto a la complejidad que cree que tiene cada tarea usando una escala de referencia como la serie de fibonacci, y posteriormente cotejar su estimación con la del resto de sus compañeros buscando establecer un valor de complejidad en consenso del equipo. En las primeras fases de ejecución del proyecto el equipo conocerá su capacidad de ejecución de puntos en un lapso de tiempo, por lo que podrá mejorar sus habilidades de estimación con base en su propia madurez.

Aspecto	Pequeño (<300 PC)		Mediano (300-600 PC)		Grande (>600 PC)	
	Empresa	**Promedio Industria**	**Empresa**	**Promedio Industria**	**Empresa**	**Promedio Industria**
Productividad (PC / Esfuerzo)	4.69	10.73	7.38	9.27	5.62	6.34
Esfuerzo (Meses Ingeniero)	39.73	18.64	72.42	43.15	196.2	126.18
Duración (Meses Calendario)	9.62	9	8.5	13	20.83	19

Tabla 1. Estimación de proyectos de software con base en puntos de complejidad (PC)

Sin embargo, considerando que la industria de software en Venezuela es una industria pequeña que está en pleno proceso de desarrollo (Rivero, 2007), es de entender que aún cuando no se cuentan con las estadísticas respectivas al caso las estimaciones de esfuerzo con puntos de función para cada tipo de proyecto en las empresas de este país son notablemente más bajas que las indicadas en el promedio de la industria. En tal sentido, los tipos de proyectos utilizados para realizar las pruebas de esta investigación se refiere a escenarios diferentes, los cuales son indicados posteriormente.

Capítulo 3. Fundamentos de Metaheurísticas

Este capítulo contiene la definición general de las metaheurísticas y algunas de sus formas más comunes de clasificación. También incluye la definición de las estrategias metaheurísticas utilizadas en este trabajo para dar solución al problema de planificación de proyectos de software: recocido simulado, búsqueda con vecindad variable y algoritmos genéticos. Finalmente, presenta los aspectos más importantes relacionados con los mecanismos de hibridación construidos a partir de la combinación de los componentes de diversas metaheurísticas.

3.1 Definición General

Tal como se ha indicado, los problemas de optimización combinatoria son de gran importancia para diversos campos de aplicación. Por tal motivo, a lo largo del tiempo se han desarrollado diversas estrategias para buscarles una solución (Chicano, 2007). Dichas estrategias, conocidas como técnicas de optimización, pueden ser clasificadas como se indica en la Figura 3.

En primer lugar, las técnicas exactas son aquellas que buscan garantizar la selección de la solución óptima para un problema determinado en un tiempo acotado (en caso de ser posible), por lo que su principal inconveniente radica en que el tiempo necesario para encontrar una solución crece exponencialmente con el tamaño del

problema, haciendo que en muchos casos estas técnicas resulten inviables. Por su parte, los algoritmos aproximados buscan encontrar una "buena" solución, sin que ésta sea la óptima. Finalmente, existen otras técnicas de optimización, por ejemplo las basadas en modelos a partir de funciones probabilísticas, según las cuales se busca seleccionar una solución de acuerdo con una probabilidad dada por una distribución determinada.

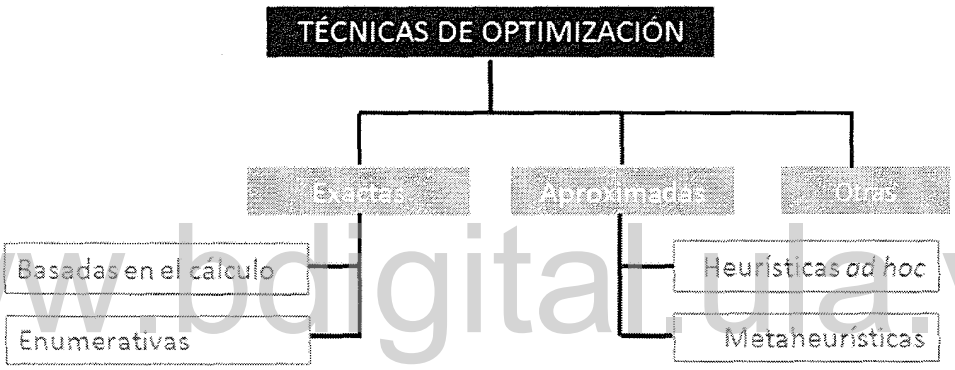


Figura 3. Clasificación de Técnicas de Optimización. Fuente: Chicano (2007)

Dentro de los algoritmos aproximados se encuentran los heurísticos constructivos y los métodos de búsqueda local. En los heurísticos constructivos se construye una solución mediante la incorporación de componentes, hasta obtener una solución completa. La calidad de las soluciones obtenidas por estos métodos por lo general depende del tipo de problema abordado y de cómo sea planteado. Por su parte, los métodos de búsqueda local parten de una solución completa y recorren el espacio de búsqueda a través del concepto de vecindad, hasta encontrar un óptimo

local, el cual es la mejor solución de su vecindad. Una vecindad es un subconjunto de soluciones, que se pueden construir aplicando un operador de modificación a partir de la solución original.

A partir de los resultados obtenidos por los distintas metaheurísticas implementadas, surge una nueva clase de algoritmos aproximados con los que se intenta mejorar la exploración del espacio de búsqueda. Estas técnicas se conocen como metaheurísticas, anteriormente conocidas como heurísticas modernas, cuyo funcionamiento se basa en la combinación de diferentes métodos heurísticos a un nivel más alto, i.e., son estrategias particulares que guían el proceso de búsqueda buscando hacerlo más efectivo y eficiente (Glover, 1986).

La implementación de metaheurísticas permite equilibrar correctamente los principios de diversificación e intensificación. Diversificar se refiere a la evaluación de soluciones en regiones distantes del espacio de búsqueda, mientras que intensificar, por el contrario, se refiere a la evaluación en regiones acotadas y pequeñas, centradas en la vecindad de soluciones concretas. El correcto balance de estos dos principios es fundamental para intensificar rápidamente las regiones prometedoras del espacio de búsqueda global, sin malgastar tiempo en las regiones que ya han sido exploradas o que no contienen soluciones de alta calidad.

Por otra parte, se debe mencionar que a pesar que la clasificación de técnicas de optimización planteada por Chicano (2007) es apropiada para introducir las

técnicas metaheurísticas, existen otras clasificaciones de técnicas de optimización como por ejemplo la planteada por Argonne National Laboratory.

3.2 Clasificación de las metaheurísticas

Existen múltiples formas de clasificar las metaheurísticas según sus características: basadas en poblaciones o en métodos de trayectoria, con memoria o sin ella, con una o varias estructuras de vecindades, etc. A continuación se describen algunos de las clasificaciones más comunes (Martínez, 2011).

Basadas en poblaciones o en métodos de trayectoria

Esta clasificación también es conocida como basada en la naturaleza, dados los principios que se pueden utilizar para obtener una solución. En primer lugar se encuentran los métodos basados en trayectoria, los cuales parten de una solución y mediante la exploración de la vecindad van actualizando la solución actual, por lo que son consideradas extensiones de los métodos de búsqueda local, exceptuando el mecanismo con el que cuentan para escapar de los mínimos locales (por ejemplo definiendo un número máximo de iteraciones sin que se obtenga ninguna mejora de la solución actual) (Martínez, 2011). Dentro de estos métodos se encuentra el recocido simulado (SA), la búsqueda tabú (TS), el procedimiento de búsqueda miope aleatorizado y adaptativo (GRASP), la búsqueda de vecindad variable (VNS), la búsqueda local iterada (ILS), entre otros.

Por su parte, los métodos basados en población a diferencia de los métodos basados en trayectoria, se caracterizan por trabajar con un conjunto de soluciones en cada iteración, por lo que se aplican un conjunto de reglas para determinar cómo generar otras soluciones a partir de dicho conjunto. Dentro de esta categoría se encuentran los algoritmos genéticos (GA), los algoritmos de estimación de distribuciones (EDA), la búsqueda dispersa (SS), la optimización basada en colonias de hormigas (ACO), la optimización basada en cúmulos de partículas (PSO), entre otros.

De acuerdo con el uso de memoria

Algunas metaheurísticas utilizan diferentes mecanismos de memoria para registrar la evolución del proceso de búsqueda, para evitar reevaluar espacio del conjunto S ya visitados. Algunas metaheurísticas que no incorporan mecanismos de memoria son el VNS, SA y GRASP. Por su parte, el TS y la optimización basada en colonia de hormigas usan mecanismos de memoria de corto y largo plazo.

De acuerdo con la estructura de vecindad

El uso de múltiples estructuras de vecindades se realiza con el fin de permitir diversificar la búsqueda de soluciones y evitar el estancamiento en óptimos locales. El VNS usa diferentes estructuras de vecindad, mientras que el SA e ILS usan una única estructura.

A continuación se describen las metaheurísticas implementadas en esta investigación para dar solución al problema de planificación de proyectos de software.

3.3 Recocido Simulado

El recocido simulado (SA por sus siglas en inglés), recibe su nombre porque su comportamiento se asemeja al proceso de recocido del acero y del vidrio, en donde inicialmente las partículas se mueven con gran rapidez y de forma desordenada, pero a medida que disminuye la temperatura buscan ordenarse entre sí, siendo los movimientos más estructurados y por tanto más ordenados (Dowsland & Adenso, 2003). Es un algoritmo de búsqueda local que parte de una solución inicial seleccionada de manera aleatoria. Seguidamente, un vecino de esta solución es generado por algún mecanismo adecuado y el cambio en el costo es calculado. Si el costo se reduce con respecto a la solución actual, ésta es reemplazada por el vecino generado (dependiendo del valor de la temperatura), de otra manera la solución se mantiene. El proceso se repite hasta que no se encuentran mejoras en la vecindad de la solución actual (mínimo local), ó cuando se llega a cierta temperatura.

Una de las principales desventajas de esta metaheurística es que el mínimo local puede estar muy lejos del mínimo global. Para ello, el algoritmo busca evitar caer en el mínimo local aceptando en algunas ocasiones vecinos que incrementan la función objetivo, es decir soluciones que son peores que la solución actual. Esta

aceptación ocurre con una probabilidad dada por la función de aceptación que es $e^{-I/T}$, en donde I es la diferencia de costo entre la solución actual y la solución generada y T es un parámetro de control que es análogo a la temperatura en el recocido físico, es decir es el valor actual de la temperatura en el algoritmo.

Para entender con mayor detalle el comportamiento de dicho algoritmo, a continuación se presenta el pseudocódigo correspondiente (Ver Figura 4), sus parámetros iniciales son:

- Valor inicial de T: Valor inicial de la temperatura
- Función de enfriamiento: función que determina en qué medida la temperatura se va enfriando progresivamente
- Numero de iteraciones N(t): cantidad de iteraciones que se van a ejecutar por cada temperatura
- Condición/Criterio de parada para terminar el algoritmo: condición de salida.

```
1  S_o: // solución inicial
2  S_φ: // mejor solución encontrada
3  S: // solución actual
4  FUNCTION SimulatingAnnealing
5  S ← S_o;
6  S_φ ← S_o;
7  t ←CalculateInitialTemperature(): // determina la temperatura
8                                    // inicial
9  while stopping criterion is not reached do
10 |  S^# ←Neighborhood(S): // La función de vecindad devuelve la mejor
11 |                       // solución vecina de S.
12 |  Δf ← f(S_φ) − f(S^#);
13 |  if Δf < 0 or Rand[0,1) < e^(−Δf/t) then
14 |  |  S ← S^#: // actualiza la solución S
15 |  end if
16 |  if f(S) < f(S_φ) then
17 |  |  S_φ ← S: // actualiza la mejor solución
18 |  end if
19 |  t ←UpdateTemperature(t): // actualiza la temperatura
20 end while
21 return S_φ
```

Figura 4. Pseudocódigo del SA

3.4 Búsqueda con vecindad variable

La búsqueda de vecindad variable (VNS por sus siglas en inglés) fue originalmente propuesta por Hansen y Mladenovic (2003). En esta metaheurística se exploran vecindades distantes de la solución actual de manera incremental, buscando moverse a partir de la solución actual si se ha obtenido alguna mejora en las vecindades próximas (Martínez, 2011). Básicamente, se fundamenta en un algoritmo de búsqueda local mejorado a partir del cambio dinámico de vecindades, bajo las siguientes premisas:

- Un óptimo local en una determinada vecindad, no necesariamente lo es en otra vecindad. Una solución es *k*-vecina de otra, si entre si existen k cambios con respecto a la estructura que las define.
- Un óptimo global es un óptimo local para todas las estructuras de vecindades.
- Para muchos problemas de optimización, los óptimos locales están relativamente entre ellos.

El funcionamiento básico del VNS tiene tres fases:

- Agitación (Shaking): a partir de la solución s´ (la cual es generada aleatoriamente) y una estructura de vecindad determinada, se busca generar una solución sobre una vecindad *k* veces más grande, lo cual conduce a una búsqueda de soluciones estocástica y donde la siguiente solución dependerá de la solución inmediatamente anterior,
- Búsqueda Local: se aplican algoritmos de búsqueda local en la vecindad correspondiente a la solución s´.
- Movimiento (Move): de acuerdo con el resultado de la evaluación de la función objetivo se decide si cambiar o no de vecindad. Si la nueva solución es mejor, se buscará una solución desde una vecindad más pequeña con miras a encontrar otra solución que mantenga buenos atributos del óptimo local (en la misma vecindad). En caso contrario, se buscará otra solución en una vecindad más grande.

```
1  S_o: // solución inicial
2  S_φ: // mejor solución encontrada
3  S: // solución actual
4  FUNCTION VariableNeighbourhoodSearch
5  S ← S_o;
6  S_φ ← S_o;
7  while stopping criterion is not reached do
8  |  k ← 1;
9  |  while k ≤ k_max do
10 |  |  S ←NEIGHBORHOOD(S, k): // La función de vecindad devuelve la
11 |  |                        // mejor solución vecina de S. Donde k
12 |  |                        // indica la k-ésima estructura de
13 |  |                        // vecindad
14 |  |  if f(S) < f(S_φ) then
15 |  |  |  S_φ ← S; // actualiza la mejor solución
16 |  |  |  k ← 1;
17 |  |  else
18 |  |  |  k ← k + 1;
19 |  |  end if
20 |  end while
21 end while
22 return S_φ
```


Figura 5. Pseudocódigo del VNS

El pseudocódigo del VNS se indica en la Figura 5. Es importante destacar que para la aplicación del VNS en el problema de planificación de proyectos de software una forma de realizar movimientos en el espacio de soluciones es cambiando algunos de los elementos que conforman la solución, lo cual permite obtener diferentes vecindades cuyo tamaño dependerá de la cantidad de cambios a realizar en dicha solución, los cuales pueden estar indicados por ejemplo por quien debe realizar cual tarea.

3.5 Algoritmos Genéticos

Los algoritmos genéticos, también conocidos como algoritmos evolutivos, se pueden definir como:

> *"Son algoritmos de búsqueda basados en la mecánica de selección natural y de la genética natural. Combinan la supervivencia del más apto entre estructuras de secuencias con un intercambio de información estructurado, aunque aleatorizado, para constituir así un algoritmos de búsqueda que tenga algo de las genialidades de las búsquedas humanas" (Goldberg, 1989)*

Es decir, para alcanzar la solución a un problema se parte de un conjunto inicial de individuos, llamado *población*, generado de manera aleatoria. Cada uno de estos individuos representa una posible solución al problema. La población evolucionará tomando como base un conjunto de pasos que simulan los esquemas propuestos por Darwin sobre la selección natural (Gestal, 2003). Para entender con mayor detalle el comportamiento de dicho algoritmo, es necesario abordar con más detalle cada uno de los pasos o etapas que lo componen (Chicano, 2007):

- **Inicialización.** Generación de los individuos que conforman la población inicial. Cada individuo representa una posible solución al problema en cuestión.
- **Evaluación Inicial.** Cada uno de los elementos de la población inicial tienen un valor con respecto a la función objetivo del problema. Este valor será el

factor de oportunidad según el cual el individuo podrá ser seleccionado o no para pasar a la siguiente generación (en la naturaleza, el equivalente sería una medida de la eficiencia del individuo en la lucha por los recursos).

- **Selección.** Se refiere a la estrategia según la cual se seleccionan los elementos que serán padres de los individuos de la siguiente generación o pasarán a formar parte de la misma directamente. Para ello existe un conjunto de métodos fundamentados en el principio de selección natural de Darwin, tal que se busca que los individuos con las mejores capacidades sean seleccionados para pasar a las siguientes generaciones. Los elementos más aptos tendrán entonces mejores valores con respecto a la función objetivo. La selección puede llevarse a cabo de acuerdo a algunos métodos como:

  a) *Selección por ruleta:* propuesta por DeJong, se le considera el método más utilizado desde el origen de los algoritmos genéticos. También es conocida como la Selección de Montecarlo. A cada uno de los individuos de la población se le asigna una parte proporcional a su factor de oportunidad en la ruleta, de tal forma que la suma de todos los porcentajes sea la unidad. Los mejores individuos recibirán una porción de la ruleta mayor que la recibida por los peores. Este método, aunque sencillo, resulta ineficiente a medida que aumenta el tamaño de la población, pues la complejidad crece en paralelo a la misma.

b) Selección por torneo: según la cual se busca realizar la selección con base a comparaciones directas entre individuos. Puede ser determinantica o probabilística. En la primera se seleccionan al azar un numero p de individuos, y de esa subselección se toma el individuo más apto para pasarlo a la siguiente generación. En el caso de la probabilística, se realiza el mismo proceso, pero en vez de seleccionar el elemento más apto, se realiza un cálculo de probabilidad para determinar que elemento tomar ofreciendo mayor peso a los elementos más aptos.

- **Operadores Genéticos:** Existen diversos operadores genéticos en la literatura, pero los más utilizados son los de cruce y mutación:

a) Cruzamiento o Recombinación: El operador de cruce intercambia información genética en los seres vivos, proceso que puede ser llevado a cabo mediante:

   - Cruce de un punto (SPX – Single Point Crossover): Se cortan los cromosomas (bits que forman cada una de las cadenas de los padres) por un punto seleccionado aleatoriamente para generar dos segmentos diferenciados en cada uno de ellos: la cabeza y la cola. Se intercambian las colas entre los individuos para generar los nuevos descendientes. De esta manera, ambos descendientes heredan información genética de los padres. Un

cruce genera entonces 2 nuevos individuos, de los que pueden ser tomados 1 o ambos de acuerdo a una probabilidad del 50% según sea el caso.

- Cruce de dos puntos (DPX - Double Point Crossover): En este caso, en vez de realizar un solo corte, se realizan dos, asegurando que se generen tres segmentos por cada padre. Un hijo se conforma entonces por el segmento central de uno de los padres y los segmentos laterales del otro padre.
- Cruce uniforme: establece que cada gen de la descendencia tiene las mismas probabilidades de pertenecer a uno y otro padre. Para implementarlo, se puede utilizar una máscara de cruce de valores binarios. Si en una de las posiciones de la máscara hay un 1, el gen situado en esa posición en uno de los descendientes se copia del primer padre. Si por el contrario hay un 0, el gen se copia del segundo padre. Para producir el segundo descendiente se intercambian los papeles de los padres, o bien se intercambia la interpretación de los unos y ceros de la máscara de cruce.

b) **Mutación**: este operador cambia al azar la información genética de los individuos buscando aumentar la diversidad de la población. Por ejemplo, intercambiando los valores de dos o más posiciones de la cadena que lo conforma ---mejor conocido como mutación

permutacional o *swap*---. Existen diferentes posibilidades para la implementación de operadores de mutación como lo muestra Larrañaga (1999). La mutación permutacional se realiza con una probabilidad *Pm*, que por lo general se encuentra entre el 1% y 5% (Goldberg, 1989).

- **Política de Reemplazo**: Se refiere al procedimiento por medio del cual se realiza el reemplazo de los individuos en la población después del proceso de selección y la aplicación de los operadores genéticos. Un algoritmo genético puede ser generacional o de estado estable. De tipo generacional se refiere al uso de una población temporal conformada por los individuos generados a partir del cruce de sus padres, o de la mutación. En este caso, la población inicial permanece intacta. En el caso de los algoritmos de estado estable se mantiene una única población de individuos, los cuales se van reemplazando a medida que avanza el algoritmo, según diversos criterios: eliminar los individuos menos aptos, eliminarlos aleatoriamente, eliminar a los padres, etc.

Finalmente, la población final bien sea la población temporal o la población inicial afectada por la política de reemplazo, contará con elementos que se espera que sean más aptos que los originales, por tanto, el elemento con mejor valor de la función objetivo representará la mejor solución posible.

Con base en lo anterior, en la Figura 6 se muestra la representación en pseudocódigo del algoritmo genético implementado para dar solución al problema

objeto de estudio de esta investigación, el cual es de tipo generacional, la selección de los padres se realizo de acuerdo con el método de la ruleta y el cruzamiento se realizó por medio de la técnica de cruce de un solo punto.

```
1  P: // conjunto de soluciones de tamaño N, i.e., {S_1, S_2, ..., S_N}
2  FUNCTION GeneticAlgorithm
3  P ← INITPOPULATION();
4  EVALUATEFITNESS(P);
5  S_φ ← GETBESTSOLUTION(P);
6  while stopping criterion is not reached do
7  |  P' ← ∅;
8  |  while i ∈ N/2 do
9  |  |  Q ←SELECTPARENTS(P); // selecciona 2 soluciones candidatas
10 |  |  if Rand[0,1) < p_crossover then
11 |  |  |  Q ← CROSSOVER(Q);
12 |  |  end if
13 |  |  for i ∈ Q do
14 |  |  |  if Rand[0,1) < p_mutation then
15 |  |  |  |  Q[i] ← MUTATION(Q[i]);
16 |  |  |  end if
17 |  |  end for
18 |  |  P' ← ADDSOLUTIONS(Q);
19 |  end while
20 |  P ← REPLACEPOPULATION(P');
21 |  S_φ ← GETBESTSOLUTION(P, S_φ);
22 end while
23 return S_φ
```

Figura 6. Pseudocódigo de un Algoritmo Genético

3.6 Algoritmos Meméticos

La hibridación de metaheurísticas es una técnica mediante la cual se puede dotar a las metaheurísticas de nuevas características basadas en otras, permitiendo así crear nuevas propuestas para el tratamiento de diversos problemas (Martínez, 2011), éstas son conocidas como algoritmos meméticos (AM). Los algoritmos meméticos

propuestos por Moscato (1989) y son procedimientos basados en población que se combinan con esquemas de búsqueda local junto con los operadores genéticos como el cruce y la mutación. Los AM han mostrado ser una técnica efectiva y rápida para diversos problemas, ya que logra un equilibrio efectivo entre el proceso de intensificación y diversificación. Además, los AM son considerados como metaheurística híbrida (Yavuz et al., 2006). En general, las metaheurísticas híbridas son propuestas apoyadas en la creencia de que la sinergia puede generar algoritmos de alto rendimiento que exploten y combinen las ventajas de las estrategias puras e individuales. En ese orden de ideas, estos algoritmos buscan combinar conceptos o características de diversas metaheurísticas, conservando la relación más estrecha con respecto a la estructura de los algoritmos evolutivos. Así, siendo que un algoritmo evolutivo se compone de un conjunto bien definido de pasos que obedece a un proceso de selección natural, es posible modificar la forma en que cada uno de dichos pasos se lleva a cabo, a partir de la combinación con otras metaheurísticas. En tal sentido, este algoritmo puede partir al igual que los algoritmos genéticos, de una población inicial de individuos, en la que cada uno de ellos representa una posible solución y a partir de la aplicación de un conjunto de operadores, se busca la obtención de nuevos individuos que posean mejores características y por ende que representen mejores soluciones.

Los algoritmos meméticos fueron explícitamente concebidos como un paradigma ecléctico y pragmático, abierto a la integración de otras técnicas, por lo

cual proporcionan un marco de trabajo apropiado para integrar en un único motor de búsqueda diferentes heurísticas provechosas (Cotta, 2007).

Desde el punto de vista de optimización, los algoritmos meméticos han mostrado ser más eficientes y efectivos (Chicano, 2007), ya que identifican soluciones de mejor calidad que los tradicionales algoritmos evolutivos aprovechando la información propia del dominio abordado brindada por los proceso de búsqueda local en concordandia con el teorema de *no free lunch* (Wolpert et al, 1997). Como resultado, los MAs han ganado amplia aceptación, en particular en problemas de optimización combinatoria bien conocidos con grandes instancias han sido resueltas óptimamente y donde otras metaheurísticas han fallado en la tarea de producir soluciones de calidad similar (Krasnogor & Smith, 2005).

Capítulo 4. Desarrollo

Este capítulo contiene la descripción de las actividades realizadas para llevar a cabo el proyecto, el cual fue ejecutado de acuerdo con las fases propuestas por la metodología de (Talbi, 2009): formulación, modelado y optimización del problema e implementación de la solución. Es importante destacar, que la metodología descrita abarca el desarrollo de algoritmos de optimización aplicado a diferentes escenarios, partiendo de un problema del mundo real formulado a través de un modelo matemático abstracto para el que se buscan implementar un conjunto de soluciones, cuyos resultados posteriormente serán comparados, determinando así si los algoritmos utilizados son aceptables o si es necesario hacer una revisión del modelo o en su defecto, de las metaheurísticas utilizadas como tal.

A continuación se explica la formulación del problema, concebida a partir del estudio del área de investigación.

4.1 Formulación del Problema

El problema de planificación de proyectos de software es un problema de optimización combinatoria en el que se desea encontrar una asignación de recursos (desarrolladores, analistas, etc.) a tareas para realizar un proyecto en un tiempo determinado, tratando de disminuir la duración o el costo del mismo, que aunque

pueden ser vistos como el mismo objetivo, también pueden ser abordados de manera independiente para ofrecer mayor flexibilidad al gerente de proyectos con base en las necesidades que éste pueda tener. Los recursos para la realización del proyecto son las personas involucradas en su desarrollo, cada una de las cuales poseen un conjunto de habilidades específicas, percibe un salario, y tiene un grado de dedicación máxima en cuanto a las actividades a realizar. Por su parte, el proyecto en cuestión posee un conjunto de tareas, cada una de las cuales requiere de una o más habilidades y de una o más personas para ser concretada. Asimismo, las tareas de un proyecto poseen un orden lógico en el que deben ser ejecutadas de acuerdo con las precedencias que existan entre ellas, de manera tal que sea posible abordar problemas con tareas que se pueden realizar en paralelo o de manera secuencial, según sea el caso. Para efectos de esta investigación, se asume que las actividades de cada proyecto son de tipo unimodo, es decir, sólo pueden ser ejecutadas de una misma forma, de manera tal que los efectos que pueda tener hacer una tarea de una manera u otro no forman parte del enfoque presentado en este proyecto.

En tal sentido, el problema requiere de dos grandes entradas. En primer lugar, lo que concierne al equipo de desarrollo: experiencia de cada desarrollador, disponibilidad y sueldo; y en segundo lugar, el conjunto de tareas que se deben realizar para concretar el proyecto, de cada una de las cuales se debe conocer su duración que puede estar indicada en unidades de tiempo o en puntos de complejidad (Kniberg, 2007), las habilidades que requiere para ser realizada, y la lista de tareas

que deben estar completamente hechas para iniciar su ejecución. La representación gráfica de dichas entradas se puede observar en la Figura 7 y 8.

Por otra parte, un objetivo igualmente importante de la planificación de proyectos de software es garantizar la calidad del producto que se debe generar. Aún cuando no existe un método inequívoco para asegurar tal fin, la asignación de tareas a las personas con mayores habilidades, experticia o mejor perfil puede ser una estrategia que contribuya al cumplimiento de esta meta, por lo que la asignación de las personas con las habilidades más idóneas para realizar una tarea determinada, también es objeto de estudio de esta investigación. Es de resaltar, que el problema de planificación de proyectos de software por su naturaleza es un problema multiobjetivo y dinámico, pero por razones de simplicidad y adecuación al escenario de prueba se abordó de manera mono-objetivo y con asignación de tareas de forma estática.

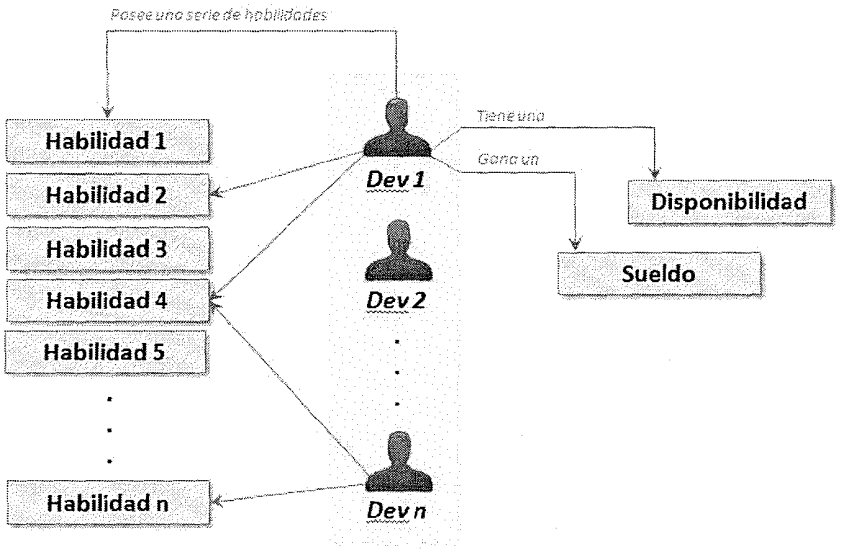


Figura 7. Estructura del equipo de desarrollo

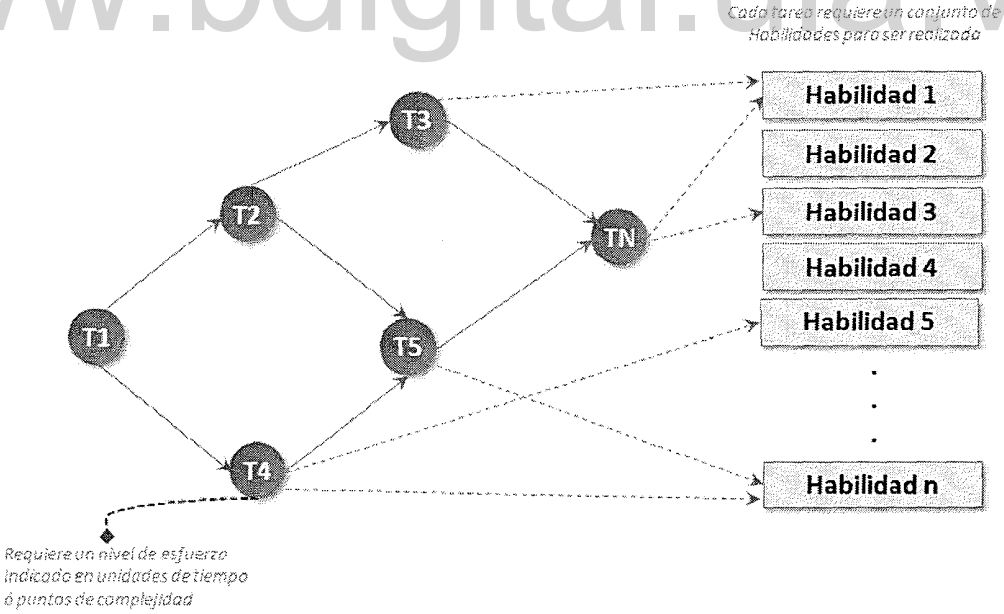


Figura 8. Estructura de las tareas de un proyecto de software

4.2 Modelado del Problema

Tal como se ha formulado el problema de planificación de proyectos de software, éste puede ser modelado a partir del PSP (Chicano, 2007), a través de un conjunto de restricciones propias del problema, como se indica a continuación.

Sea $E = \{E_1, E_2, \dots, E_n\}$ el conjunto de empleados, donde n es el número máximo de empleados y sea $H = \{H_1, H_2, \dots, H_m\}$ el conjunto total de habilidades requeridas por las tareas del proyecto y las propias de cada empleado, donde m es el número máximo de habilidades, tal que $\forall E_i,\ \exists d_i \mid d_i > 0, i = 1, 2, \dots, n$, con d_i representando la disponibilidad del i-ésimo empleado (horas/día) y $\forall E_i, \exists s_i \mid s_i > 0,\ i = 1, 2, \dots, n$; y donde s_i representa el sueldo del i-ésimo empleado medido como unidad monetaria/hora. Además $\forall E_i,\ \exists Z_i \mid Z_i = \{H_\tau, H_{\tau+1}, \dots, H_{\tau+h-1}\},\ |Z_i| = h$ con $H_{\tau+\mu} \in H$ para $\mu = 0,1, \dots, h-1$, donde Z_i representa el subconjunto de habilidades asociadas al empleado E_i.

Por otra parte, sea $T = \{T_1, T_2, \dots, Tp\}$ el conjunto total de tareas asociadas a un proyecto, donde p es el número máximo de tareas de un proyecto, tal que $\forall T_j, \exists g_j \mid g_j > 0, j = 1, 2, \dots, p$ donde g_j representa la duración de la j-ésima tarea (medida en horas). Así,

$$g_j = \beta p_j \qquad (1)$$

Donde p_j son los puntos asignados mediante la metodología *poker planning* y β es un indicador de la experticia del equipo de trabajo, así entre mayor sea β menor la experticia.

Además, $\forall T_j, \exists Z'_j \mid Z'_j = \{H_{\tau'}, H_{\tau'+1}, \dots, H_{\tau'+h'-1}\}, |Z'_j| = h'$ con $H_{\tau'+\mu} \in H$ para $\mu = 0,1, \dots, h'-1$ y donde Z'_j representa el conjunto de habilidades requeridas por la tarea T_j.

Las tareas y las precedencias existentes entre las tareas se indican mediante un grafo dirigido (TPG) $A = T \times T$. A partir del recorrido topológico del grafo se obtienen los niveles del grafo (según los cuales es posible conocer cuáles tareas deben ser realizadas antes que otras para efectos de respetar las precedencias entre las mismas) definidos como $L = \{L_1, L_2, \dots, L_\alpha\}$, donde α corresponde al número de niveles del grafo, así $L_{\varphi=}\{T_\varphi^1, T_\varphi^2, \dots, T_\varphi^f\}$ con $\varphi = 1,2, \dots, \alpha$ y $T_\varphi^w \in T$, tal que $\sum_{\varphi=1}^{\alpha} |L_\varphi| = p$.

El tiempo máximo por nivel es dependiente tanto de la dedicación de los empleados como de las tareas involucradas, por lo tanto, el lapso de tiempo necesario para llevar a cabo todas las tareas de dicho nivel, se puede definir de la siguiente manera:

$$\delta_\varphi = \max\left\{\sum_{j=T_\varphi^1}^{T_\varphi^f} X_{ij}\theta_{ij}\right\} \qquad \forall i, tal\ que\ \theta_{ij} = {}^{g_j}/_{d_i}$$

donde $i = 1, 2, \ldots . n$ y δ_φ es el tiempo máximo para el nivel φ y X_{ij} es una matriz $E \times T$ tal que:

$$X_{ij} = \begin{cases} 1 & si\ E_i\ hace\ la\ tarea\ T_j \\ 0 & de\ otra\ forma \end{cases} \quad (2)$$

De esta manera, para el cálculo del tiempo y costo de la solución se tiene:

$$f_{tiempo} = \min\left\{\sum_{\varphi=1}^{\alpha} \delta_\varphi\right\} \quad (3)$$

Donde $\forall\, T_\varphi^\omega, \exists\, T_{\varphi'}^{\omega'} \mid \left(T_\varphi^\omega, T_{\varphi'}^{\omega'} \in A\right)$ con $\varphi = 1, \ldots, \alpha - 1\ y\ \omega = 1, \ldots, |L\varphi|$

$$f_{costo} = \min\left\{\sum_{j=1}^{p}\sum_{i=1}^{n} X_{ij} g_j s_i\right\} \quad (4)$$

Adicionalmente, el universo de posibles soluciones estará conformado sólo por soluciones factibles, las cuales son aquellas soluciones que cumplen con las siguientes restricciones:

Restricción 1: Cada tarea tiene al menos una persona asignada

$$\sum_{j=1}^{p} X_{ij} > 0, \qquad j = 1,2, \ldots, p \quad (5)$$

Restricción 2: No existen tareas que requieran habilidades que no tenga ninguno de los empleados disponibles

$$X_{ij} = \left[Z'_j = Z'_j \cap Z_i\right] \quad (6)$$

Donde [·] es el paréntesis Iverson (es decir, [P] = 1, si P es cierta, y [P] = 0 en caso contrario.

Restricción 3: A ningún empleado se le asigna más tiempo de lo que su dedicación le permite.

$$\forall i,j \quad X_{ij} = 1 \quad | \quad \theta_{ij} > 1 \wedge g_j > d_i \tag{7}$$

De esta manera, la calidad de una solución se obtiene al evaluar en conjunto la duración, costo y factibilidad de la planificación generada con base en la definición formal indicada en esta sección, entendiendo que una solución es mejor que otra con base en la medida en que satisfaga las prioridades del gerente de proyectos, ya sea disminuyendo el costo o la duración del proyecto en cuestión. Es importante destacar que el riesgo que puede tener la planificación de un proyecto de software para llevarse a cabo, dado por la plataforma de hardware o software necesaria para concretar las tareas del proyecto o por las relaciones y comunicación entre los posibles desarrolladores no es considerado como parte de esta investigación.

4.3 Optimización del Problema

Aún cuando el modelo del PSP representa una base robusta para la implementación del problema de planificación de proyectos de software, resulta necesario optimizarlo a partir de las particularidades de este caso de estudio en el que igualmente se cuenta con una cantidad de empleados cada uno de los cuales posee un

conjunto de habilidades específicas, un sueldo y un grado máximo de dedicación. Asimismo, el proyecto de software cuenta con una cantidad de tareas, que requieren cierto tipo de habilidades y deben ser ejecutadas en un orden determinado para la consecución de los objetivos planteados. Sin embargo, las metodologías ágiles para el desarrollo de software y las tendencias actuales para la planificación de proyectos de software sugieren que el desglose de tareas debe llevarse a cabo de manera tal que ninguna tarea requiera más de 16 horas de trabajo por desarrollador/recurso, así como también, que cada tarea debe ser realizada en la medida de lo posible por una sola persona para evitar los mecanismos de reasignación de actividades que puedan afectar el progreso regular del proyecto (Abdel & Madnick, 1983). Además, el esfuerzo que requieren las tareas del proyecto de software puede ser indicado en puntos de complejidad o directamente en el tiempo proporcional a los mismos.

La estimación por puntos de complejidad es una técnica propia del campo de las metodologías ágiles para el desarrollo de software, en la que se indica complejidad de la tarea con respecto al tiempo que se requiere para ejecutarla, mediante la asignación de número enteros de acuerdo con la estimación del gerente de proyectos (Kniberg, 2007) ---en adelante GP, y del equipo de desarrollo. Es decir, para un GP una tarea de complejidad 5 puede indicar que se requieren 10 horas para ser resueltas, pero para otro GP indica que se requieren 5 horas. Esta estimación con base en puntos se realiza con la finalidad de aumentar la escalabilidad de la estimación del gerente de proyectos. Así, si su estimación de tiempo es incorrecta,

solo debe ajustar la proporción de puntos con respecto al tiempo (a través del coeficiente β expresado en la ecuación 1, en nuestro caso $\beta = 1$), en vez de ajustar el tiempo de cada tarea, además puede indicar tantos grados de complejidad como las tareas del proyecto lo ameriten. La proporción de puntos con respecto al tiempo podrá ser más acertada en la medida en que el equipo de desarrollo sea más maduro.

Por otra parte, para esta adaptación del modelo solo se modifica la primera restricción: cada tarea debe ser realizada por solo una persona, por la naturaleza de los proyectos en cuestión. Sin embargo, se cuenta con una restricción adicional, no obligatoria, en la que el gerente de proyectos puede indicar si desea o no maximizar la asignación de tareas a los empleados con mayor experiencia para cada una de ellas, es decir, los empleados cuyas habilidades estén más acordes con las requeridas por cada tarea, buscando utilizar una estrategia que puede tener repercusiones directas en la calidad del producto a generar. Dicho grado de experticia se define como un factor particular para cada habilidad, así para cada Z_i se tiene un $\vartheta_i = \{\vartheta_i, \dots, \vartheta_{i+h-1}\}$ con $0 \leq \vartheta_{i+r} \leq 1$ para $r \leq h-1$, donde $\vartheta_i Z_i$ son los grados de experticia para cada habilidad del empleado E_i. Cada tarea de un proyecto requiere un conjunto mínimo de habilidades $\vartheta'_j = \{\vartheta'_j, \dots, \vartheta'_{j+h'-1}\}$ con $0 \leq \vartheta'_{j+r} \leq 1$ para $r \leq h'-1$ dado que $\vartheta'_j Z'_j$ es la habilidad mínima requerida para hacer una tarea, entonces

$$X_{ij} = \left[Z'_j = Z'_j \cap Z_i \wedge Z'^{q'}_j = Z^q_i \wedge \vartheta'^{q'}_j \geq \vartheta^q_i\right] \quad \textbf{(8)}$$

donde $Z_{j'}^{q'}$ es la q´-ésima habilidad de la tarea T_j y [•] es el paréntesis Iverson (es decir, [P] = 1, si P es cierta, y [P] = 0 en caso contrario).

La solución del problema viene indicada de igual manera por una matriz en la que se especifica el grado de dedicación del empleado a cada tarea. Asimismo, la calidad de la solución se evaluará con base en la definición de la función objetivo. Para este caso particular existen dos factores fundamentales en dicha función: la duración del proyecto y el costo del mismo. En tal sentido, las metaheurísticas a implementar para ofrecer soporte al problema de planificación de proyectos de software serán desarrolladas de manera tal que el GP pueda seleccionar cual objetivo desea abordar ó ambos en caso de resultar factible, e incluso indicar el grado de especialización de cada empleado para cada habilidad requerida, con miras a diferenciar de esta manera entre desarrolladores expertos y principiantes.

Finalmente, la factibilidad de la solución no será considerada como un factor relacionado con la calidad, pues solo soluciones factibles son consideradas aptas para ser evaluadas. De esta manera, aquellas soluciones en donde los empleados deban dedicar más tiempo que lo que su dedicación les permita, o en las que una tarea deba ser realizada por más de una persona, no forman parte del posible universo de soluciones. Lo mismo ocurre para aquellas soluciones en las que existan tareas que no han sido asignadas a ninguna persona.

4.4 Implementación de la solución

A partir del modelado del problema, es posible proceder a la implementación de su respectiva solución la cual fue llevada a cabo a partir de la metodología de desarrollo incremental e iterativo (Kniberg, 2007), teniendo en cuenta que cada iteración representó la revisión de la versión más reciente del sistema así como la implementación de nuevas funcionalidades de acuerdo con los requerimientos previamente definidos, permitiendo codificar de manera progresiva las diferentes metaheurísticas seleccionadas así como también realizar las pruebas necesarias sobre las mismas.

A continuación se realiza la explicación de un ejemplo práctico en el que se busca reducir el tiempo de desarrollo del proyecto, el cual permite observar cada una de las etapas que se realizan de manera previa a la ejecución de cada una de las metaheurísticas a manera de preparación del entorno de trabajo.

En primer lugar, se presenta la descripción de las tareas necesarias para realizar el proyecto (ver la Tabla 2.), en la que se puede observar la descripción de las tareas y su respectiva duración, además de la lista de las habilidades que cada una de ellas requiere para ser completada. Esta lista es un subconjunto de la lista de habilidades generales del proyecto indicada en la Tabla 3.

N°	Descripción	Duración (g_j)	Hab. Requerida (Z'_j)				
			1	2	3	4	5
1	Ejecutar el script de la BD	4	-	-	-	-	Si
2	Implementar el diseño	4	Si	Si	Si	-	-
3	Validar el inicio de sesión	8	-	-	-	Si	-
4	Validar el registro de usuario	1	-	-	-	Si	-
5	Validar el módulo de notificaciones	3	-	-	-	Si	-
6	Instalar el sistema en producción	7	Si	Si	Si	-	-

Tabla 2. Lista de tareas *T* de un proyecto de software

N°	Descripción
1	Conocimientos de HTML
2	Conocimiento de AJAX
3	Conocimiento de Javascript
4	Programación en PHP
5	Modelado de bases de datos

Tabla 3. Lista de habilidades *H* requeridas para hacer un proyecto de software

Seguidamente se encuentra la lista de desarrolladores que conforman el grupo de desarrollo que se encargará del proyecto. De cada desarrollador se tiene su respectiva identificación, disponibilidad y el subconjunto de habilidades que posee, las cuales igualmente deben estar contenidas en la lista general de habilidades indicadas en la Tabla 3. La descripción de los desarrolladores se indica en la Tabla 4. Tal como se especificó en la formulación del problema, aquellas soluciones en las que las tareas requieren habilidades que no se encuentran en la lista general de las

mismas, se consideran inviables, así como también las habilidades adicionales que posean los desarrolladores no representan ningún valor agregado con respecto a las habilidades requeridas por cada tarea, respectivamente.

N°	Nombres y Apellidos	Disponibilidad (d_i)	Sueldo(s_i)	Habilidad que posee (Z_i)				
				1	2	3	4	5
1	Pedro Pérez	Tiempo Completo (8)	150	Si	Si	Si	Si	Si
2	María Ramírez	Tiempo Completo (8)	50	Si	Si	Si	Si	-
3	Juan Parra	Tiempo Completo (8)	100	Si	Si	Si	Si	Si

Tabla 4. Lista de integrantes de un equipo de desarrollo *E* para proyectos de software

Para completar los datos de entrada necesarios para comenzar la búsqueda de una solución, solo resta conocer las precedencias entre las tareas que conforman el proyecto, las cuales vienen dadas por el TPG indicado en la formulación del problema tal como se puede observar en la Figura 9.

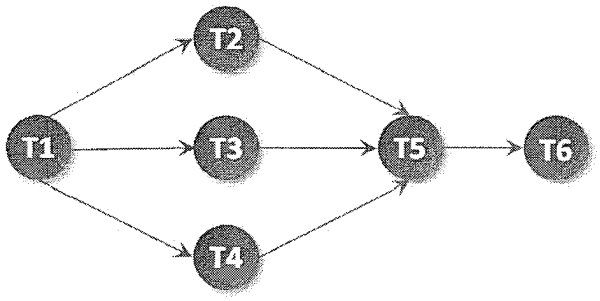


Figura 9. Grafo de Precedencia de tareas *A* de un proyecto de software

Una vez conocidas todas las variables involucradas en la representación del problema, el primer paso consiste en determinar la factibilidad de búsqueda de una solución, cumpliendo así con la primera y segunda restricción del modelo. Para ello, se realiza una búsqueda en la que se indica quién puede realizar cuáles tareas, tal como se indica en la Tabla 5. En el caso en el que exista una tarea que no pueda ser realizada por ninguno de los integrantes del equipo de desarrollo de acuerdo con las habilidades que la misma requiere, automáticamente se considera que no es factible realizar la búsqueda de una solución determinada. Adicionalmente, a partir de este procedimiento es posible conocer quiénes pueden llevar a cabo cada una de las tareas, conformando así el universo de posibles soluciones.

N°	Nombres y Apellidos	Tarea que puede realizar					
		1	2	3	4	5	6
1	Pedro Pérez	1	1	1	-	1	
2	María Ramírez	-	-	1	1	-	1
3	Juan Parra	-	-	-	1	1	1

Tabla 5. Desarrolladores candidatos para cada tarea del proyecto representado por la matriz X_{ij}

Posterior a la verificación de factibilidad del proyecto a evaluar, es necesario determinar el orden en el que las tareas pueden ser ejecutadas considerando aquellos niveles del grafo conformados por tareas que se pueden realizar paralelamente, y que por ende tendrán una repercusión directa en la duración de la solución a generar. Para ello, se realiza un recorrido topológico del grafo (ver Anexo A) mediante el cual se

determina a qué nivel pertenece cada uno de sus nodos, es decir cada una de las tareas del proyecto.

Finalmente, se procede a generar las respectivas soluciones. Al aplicar la ecuación 2 en la solución S_1 indicada en la Tabla 6, se obtiene que se requieren 26 horas para culminar el proyecto, equivalente a 1.08 días laborales considerando que todos los desarrolladores involucrados poseen disponibilidad de tiempo completo (se asume que la jornada laboral dura 8 horas). Por su parte, en la planificación indicada en la Tabla 7 correspondientes a una segunda solución S_2, se requieren 22 horas de trabajo, valor obtenido a partir de la misma ecuación, equivalente a 0.91 días laborales (Ver Ecuación 2)

N°	Nombres y Apellidos	Responsable de la tarea					
		1	2	3	4	5	6
1	Pedro Pérez	Si	Si	-	-	Si	-
2	María Ramírez	-	-	Si	-	-	Si
3	Juan Parra	-	-	-	Si	-	-

Tabla 6. Solución S1 – Planificación de proyecto de software

N°	Nombres y Apellidos	Responsable de la tarea					
		1	2	3	4	5	6
1	Pedro Pérez	1	1	1	-	-	-
2	María Ramírez	-	-	-	1	-	-
3	Juan Parra	-	-	-	-	1	1

Tabla 7. Solución S2 - Planificación de proyecto de software

Por otra parte, la implementación de la solución de acuerdo con este ejemplo, ofreció la flexibilidad necesaria para permitir que el gerente de proyectos indique el grado en el que cada tarea requiere una determinada habilidad, así como también el grado en el que un desarrollador la posee. Este grado se indica como un valor numérico entre 0 y 1, siendo 1 el límite de mayor experiencia posible de una habilidad dada. De esta manera se puede obtener un coeficiente que indique qué tan buen candidato resulta un determinado desarrollador para una tarea en particular, para efectos de dar respuesta a aquellos casos de planificación en los que se busca asignar a los desarrolladores con mayores conocimientos o experticia a cada tarea respectivamente. Por ejemplo, de acuerdo con la información indicada en las Tablas 8 y 9, pueden generarse diversas planificaciones en las por ejemplo el desarrollador 1 resulta mejor candidato que el desarrollador 3 para hacer la tarea 5, mientras que el desarrollador 2 tiene mejores capacidades para hacer la tarea 4 que el desarrollador 3.

Una vez completado los pasos de evaluación del proyecto y la factibilidad de la solución a buscar, se procede a la implementación de cada una de las metaheurísticas de acuerdo con la función objetivo a evaluar, ya sea de reducción de costos o reducción de tiempo de acuerdo con las variantes de sueldos y disponibilidades de cada una de los empleados que integran el equipo de desarrollo como tal.

N°	Descripción	Duración (g_j)	Hab. Requerida (Z'_j)				
			1	2	3	4	1
1	Ejecutar el script de la BD	4	-	-	-	-	1
2	Implementar el diseño	4	0.5	0.5	0.7	-	-
3	Validar el inicio de sesión	8	-	-	-	1	-
4	Validar el registro de usuario	1	-	-	-	0.5	-
5	Validar el módulo de notificaciones	3	-	-	-	0.5	-
6	Instalar el sistema en producción	7	0.5	1	0.2	-	-

Tabla 8. Lista de tareas *T* de un proyecto de software – Grados por habilidad

N°	Nombres y Apellidos	Disponibilidad (d_i)	Sueldo (s_i)	Habilidad que posee (Z_i)				
				1	2	3	4	5
1	Pedro Pérez	Tiempo Completo (8)	150	0.5	0.5	0.5	1	1
2	María Ramírez	Tiempo Completo (8)	50	1	0.5	0.2	0.5	-
3	Juan Parra	Tiempo Completo (8)	100	1	1	1	0.1	0.5

Tabla 9. Lista de desarrolladores *E*– Grados por habilidad

A continuación se detalla la implementación de cada una de las metaheurísticas seleccionadas para abordar el problema objeto de investigación, correspondientes a métodos sin memoria de trayectoria y de población.

4.5 Implementación del SA

La implementación del SA se realizó con base en el pseudocódigo indicado en la Figura 4. La solución inicial es generada aleatoriamente de acuerdo con los posibles candidatos para una tarea determinada. Dicha solución posee un valor de la función objetivo en cuanto a tiempo o costo de desarrollo, o coeficiente de experticia según sea el objetivo de la planificación a generar por parte del gerente de proyectos. La siguiente solución, nuevamente es generada aleatoriamente, considerando para todos los casos las restricciones de posibles cambios, y se realiza la comparación respectiva entre los valores de la función objetivo. En caso de que la nueva solución sea mejor que la anterior, ésta pasa a ser la actual, y por ende, el siguiente punto de comparación.

Ahora bien, dado que una de las principales desventajas de esta metaheurística, es que el mínimo local puede estar muy lejos del mínimo global, se establece la posibilidad de aceptar en algunas ocasiones soluciones peores que la actual, lo cual ocurre con una probabilidad dada por la función de aceptación que es $e^{-I/T}$, en donde I es la diferencia de costo, tiempo o coeficiente de experticia de la solución actual y la solución generada, y T es un parámetro de control que es análogo a la temperatura en el recocido físico, es decir es el valor actual de la temperatura en el algoritmo.

4.6 Implementación del VNS

Para la implementación del VNS, es necesario en primer lugar conocer las estructuras de las vecindades que puede tener una instancia del problema en particular. En el ejemplo indicado en la Tabla 5, la vecindad más grande que puede tener la solución inicial generada de manera aleatoria sería de tamaño 6, pues cada una de las tareas puede ser desarrollada por más de una persona. Sin embargo, cuando se considera el conjunto de habilidades que posee cada desarrollador versus el conjunto de habilidades que requiere cada tarea, esto no es necesariamente cierto, pues para aquellos casos en los que una tarea solo pueda ser realizada por una persona no es posible realizar cambios en dicha asignación por lo que esa vecindad no es factible. Por tanto, la vecindad más grande que puede tener un problema viene dada por el total de tareas que se tienen, menos aquellas tareas que pueden ser realizadas solo por una persona.

A partir de este estudio de las estructuras de las vecindades, se procede a ejecutar el algoritmo tal como se indica en la Figura 5, generando la primera solución de manera aleatoria con el respectivo cálculo de su *fitness* en función del tiempo o costo del proyecto según sea el caso. Posteriormente, se evalúa otra solución de la misma vecindad y se comparan los resultados. En caso de ser mejor se continúa en la misma estructura para intensificar la búsqueda, y en caso contrario se pasa a una estructura más grande para diversificar la evaluación del espacio de soluciones. La

condición de parada de este algoritmo se indico como máxima cantidad de iteraciones sin mejora sobre la cantidad máxima de evaluaciones de la función objetivo.

4.7 Implementación del GA

El algoritmo genético implementado en esta investigación es de tipo generacional, la selección de los padres se realizó de acuerdo con el método de la ruleta y el cruzamiento se realizó por medio de la técnica de cruce de un solo punto. Por su parte, se establecieron tres escenarios del GA a partir de la variación de su probabilidad de mutación indicada como 5, 10 y 15% respectivamente, mientras que la probabilidad de recombinación utilizada fue del 100%. El pseudocódigo correspondiente a este método se indica en el Figura 6.

Es importante resaltar, que los individuos de la población inicial, 30 en total, fueron generados de manera aleatoria, calculando para cada uno de ellos su correspondiente valor en cuanto a duración y costo de la planificación generada. Por cada ejecución del algoritmo se permite la evolución de 10 generaciones, siendo la última generada la población que se evalúa para seleccionar la solución definitiva. De esta manera, para los escenarios de prueba con mayores combinaciones posibles la evolución de las generaciones ofrece una mejor estrategia para recorrer el espacio de búsqueda.

Para todas metaheurísticas implementadas en el proyecto, los diferentes operadores definidos (vecindad, cruce, mutación, etc.) manejan solo soluciones

factibles y se calcula la función objetivo de estas soluciones mediante la Ecuación 3 o 4, según sea el caso (tiempo o costos). Para la verificación del cumplimiento de las restricciones se utiliza una estructura de datos matricial denominada W_{ij} y equivalente estructuralmente a X_{ij}, donde se indica cuales tareas puede o no realizar un determinado empleado. Lo anterior se logra mediante la definición de una función denominada *feasible(E,T)* al inicio de cada metaheurística. Así por ejemplo, para el caso del SA, el operador para la generación de una nueva solución parte de W_{ij} para generar aleatoriamente una X_{ij}, de forma tal que se logran dos beneficios. El primero, se garantiza la existencia de soluciones factibles y el segundo, se reduce el esfuerzo computacional en la búsqueda de soluciones factibles.

4.8 Interfaz Gráfica de Entrada y Salida

Tal como se ha indicado, los datos de entrada para generar las planificaciones de los proyectos de software en cuestión se refieren a: la lista de habilidades que pueden tener los desarrolladores y requerir las tareas, y las listas de tareas y desarrolladores con sus datos respectivamente, así como también el grafo de precedencia de tareas. La interfaz de entrada (ver Figura 10) le permite al usuario indicar la ruta en donde se encuentran los archivos correspondientes a dichos datos, así como también la configuración de los valores de cada metaheurística. Los archivos de entrada son leídos como archivos separados por comas (CSV por sus siglas en inglés) con la estructura indicada en las figuras 11, 12 y 13. La aplicación

fue desarrollada en Java 1.6, utilizando un equipo con un procesador Pentium dual-core 2.60 GHz, 4 GB de RAM y sistema operativo de 64 bits para la codificación y ejecución de las pruebas.

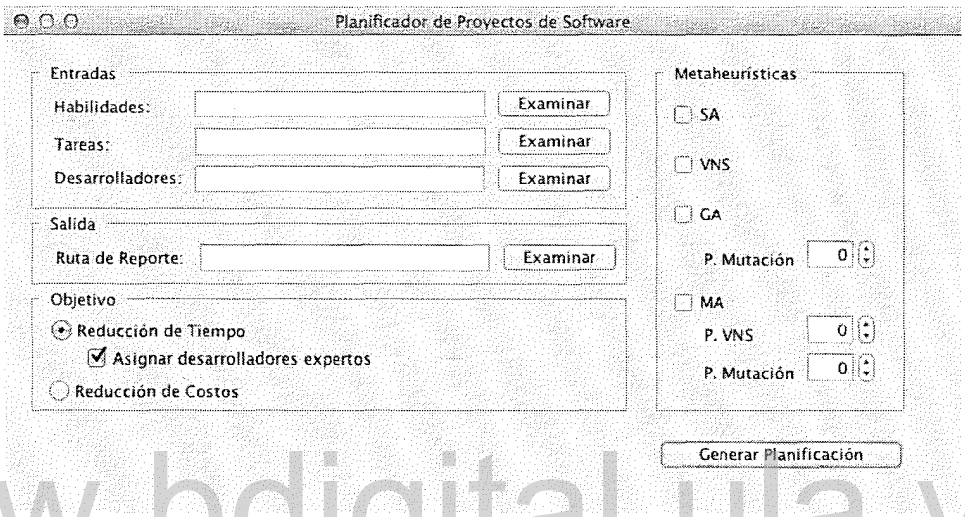


Figura 10. Interfaz de entrada

Skills.csv

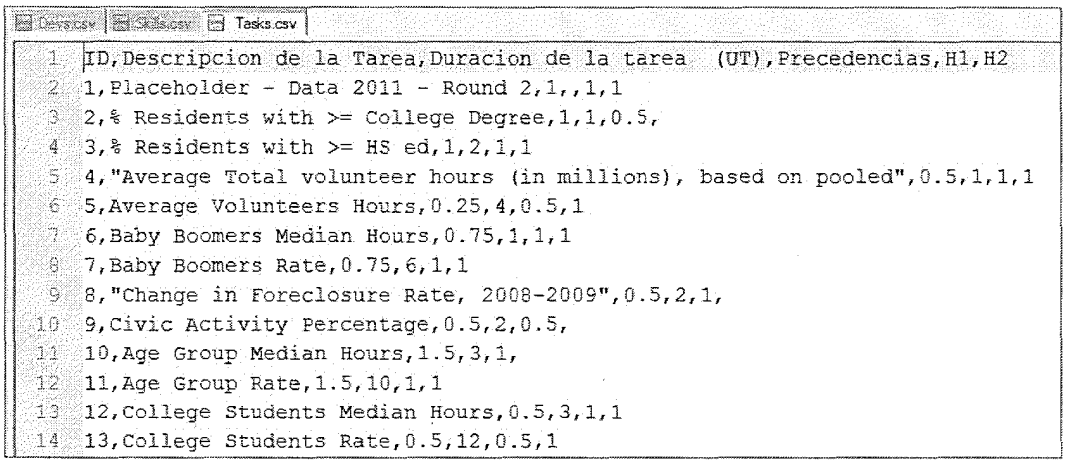
```
1  ID,Descripcion de la Habilidad,Seniority
2  1,Web Programming,1
3  2,Database Manager,1
```

Figura 11. Estructura del archivo de entrada de Habilidades

Devs.csv

```
1  ID,Nombre Completo,Disponibilidad,Pago por hora,H1,H2
2  1,Pedro Perez,1,50,1,
3  2,Maria Ramirez,1,100,0.5,1
```

Figura 12. Estructura del archivo de entrada de Desarrolladores



Tasks.csv

```
1  ID,Descripcion de la Tarea,Duracion de la tarea  (UT),Precedencias,H1,H2
2  1,Placeholder - Data 2011 - Round 2,1,,1,1
3  2,% Residents with >= College Degree,1,1,0.5,
4  3,% Residents with >= HS ed,1,2,1,1
5  4,"Average Total volunteer hours (in millions), based on pooled",0.5,1,1,1
6  5,Average Volunteers Hours,0.25,4,0.5,1
7  6,Baby Boomers Median Hours,0.75,1,1,1
8  7,Baby Boomers Rate,0.75,6,1,1
9  8,"Change in Foreclosure Rate, 2008-2009",0.5,2,1,
10 9,Civic Activity Percentage,0.5,2,0.5,
11 10,Age Group Median Hours,1.5,3,1,
12 11,Age Group Rate,1.5,10,1,1
13 12,College Students Median Hours,0.5,3,1,1
14 13,College Students Rate,0.5,12,0.5,1
```

Figura 13. Estructura del archivo de entrada de Tareas

Por su parte, una vez generada la planificación del proyecto en cuestión, esta es mostrada al gerente de proyectos en una página web de formato HTML, que contienen los siguientes componentes:

- **Descripción de la Solución:** en este cuadro se indica el resumen de la planificación generada para el proyecto, la cual corresponde a la solución definitiva del método utilizado. Se indica duración del proyecto, costo, algoritmo utilizado, objetivo seleccionado y el tiempo de ejecución que se emplea para generar la planificación calculado a partir de la diferencia entre la hora del sistema cuando inició y finalizó el método respectivamente (ver Figura 15).
- **Descripción de la planificación:** corresponde a una tabla detallada en donde se indica el responsable de cada una de las tareas del proyecto (ver Figura 16)

- **Descripción de la planificación:** corresponde a una tabla detallada en donde se indica el responsable de cada una de las tareas del proyecto (ver Figura 16)
- **Grafo de precedencia de tareas:** en el cual se expresan gráficamente las precedencias entre las tareas, para permitirle al gerente de proyectos observar con mayor facilidad los puntos que pueden resultar como cuellos de botella durante la ejecución del proyecto. En la Figura 17 se observa una vista parcial de un grafo de precedencia de tareas, el cual no se muestra completamente por motivos de espacio.
- **Carga de trabajo por desarrollador:** es una gráfica de barras en donde se indica la proporción del trabajo asignado a cada uno de los desarrolladores involucrados (ver Figura 18)
- **Costos por desarrollador:** de igual manera corresponde a un gráfico de barras que indica la proporción de costos que genera cada desarrollador de acuerdo con la planificación generada Figura 19. Esta gráfica en conjunto con la carga por desarrollador le permite al gerente tener una visión más clara acerca de la asignación de recursos.
- **Planificación en función del tiempo:** es una representación del cronograma de actividades durante todo el lapso de tiempo necesario para concretar el desarrollo del proyecto, en la que se indica por cada unidad de tiempo el estado de ocupación de cada desarrollador (ver Figura 20).

Descripción del Problema

Cantidad de Tareas:	131
Cantidad de Desarrolladores:	2
Cantidad de Habilidades:	2

Figura 14. Interfaz de Salida – Descripción del Problema

Descripción de la Solución

Duración:	Aprox. 104.45 dias
Costo:	11972.5 U.M.
Algoritmo:	GA
Objetivo:	Reducción de Costos
Tiempo de Ejecución:	47.0 seg.

Figura 15. Interfaz de Salida – Descripción de la Solución

Tarea	Duración (U. T.)	Asignada a
1	1.0	Maria Ramirez
2	1.0	Pedro Perez
3	0.5	Maria Ramirez
4	0.75	Maria Ramirez
5	1.0	Maria Ramirez
6	0.5	Maria Ramirez
7	0.5	Maria Ramirez
8	0.25	Maria Ramirez
9	0.75	Maria Ramirez
10	1.5	Pedro Perez
11	0.5	Maria Ramirez
12	0.5	Maria Ramirez
13	0.5	Pedro Perez
14	0.5	Pedro Perez
15	0.5	Maria Ramirez
16	2.0	Pedro Perez
17	0.5	Maria Ramirez
18	0.5	Maria Ramirez
19	0.5	Maria Ramirez
20	0.5	Maria Ramirez
21	0.5	Maria Ramirez
22	1.5	Maria Ramirez
23	0.5	Pedro Perez
24	1.25	Maria Ramirez

Figura 16. Interfaz de Salida – Descripción de la Planificación

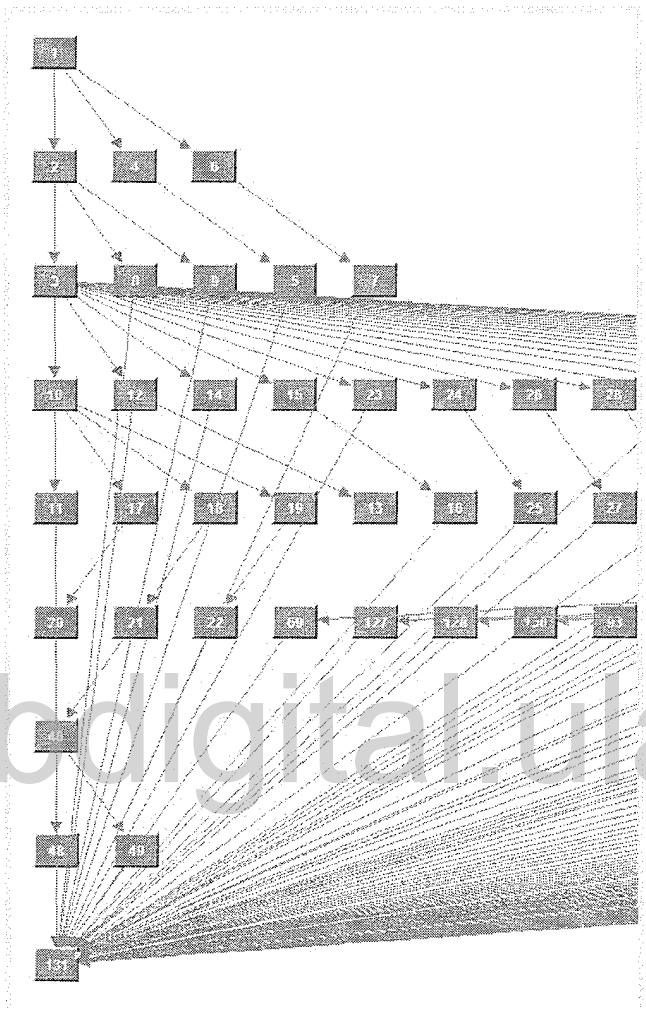


Figura 17. Interfaz de Salida – Vista Parcial del Grafo de precedencia de tareas (TPG)

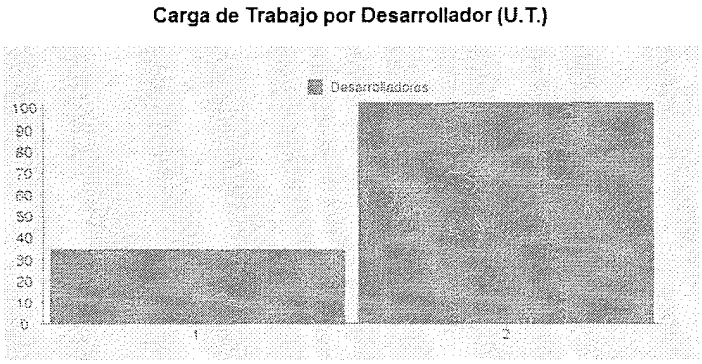


Figura 18. Interfaz de Salida – Carga de trabajo por desarrollador

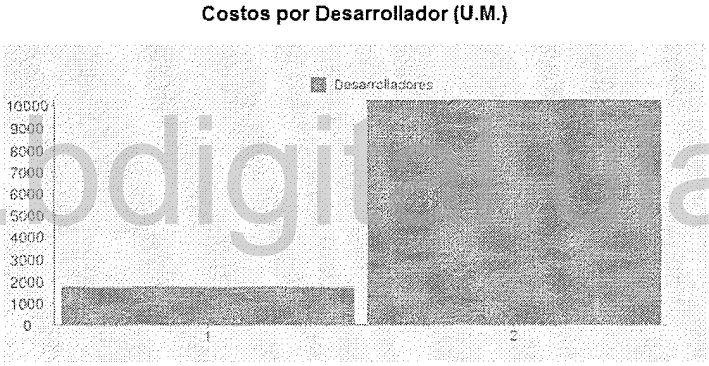


Figura 19. Interfaz de Salida – Costos por desarrollador

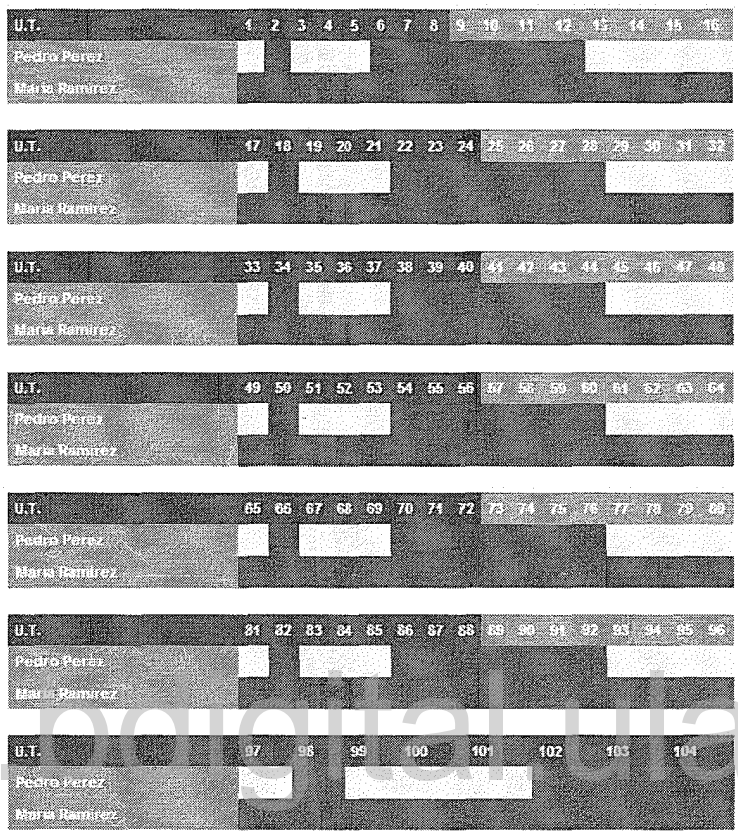


Figura 20. Interfaz de Salida – Planificación en función del tiempo

Capítulo 5. Resultados

5.1 Definición de casos de prueba

Para la verificación del funcionamiento y rendimiento de las metaheurísticas implementadas, se realizó un conjunto de pruebas sobre distintos escenarios. Cada escenario corresponde a un tipo de proyecto de software, los cuales fueron clasificados de acuerdo a la cantidad de puntos acumulada a través de sus respectivas tareas, tal como se indica en la Tabla 10, con base en la evaluación de la magnitud de los proyectos de software de la empresa Caniatech C. A (Caniatech, 2012), correspondiente a una empresa local dedicada al desarrollo de diversos tipos de aplicaciones web. La definición de los tipos de proyectos con base en esta clasificación, se realizó en primer lugar por la ausencia de una clasificación estándar de esta área en lo que a materia de proyectos de software se refiere, y en segundo lugar para dar paso a futuras investigaciones relacionadas con la metodología aquí desarrollada y cotejarla con los resultados obtenidos por la empresa a partir de las soluciones propuestas. Los casos de prueba evaluados se indican en la Tabla 11.

Por otra parte, es importante mencionar que si bien existen *benchmarks* para problemas de planificación como por ejemplo OR-LIBRARY, la única referencia sobre potenciales instancias asociadas al problema de planificación de software fue dado por Chicano (2007), pero su propuesta está orientado a la generación de

instancias de prueba y no de *benchmark* estático. Por otra parte, se tenía como meta probar con instancias del problema brindadas por la experiencia de la compañía Caniatech en este rublo.

Tipo	Descripción
Pequeño	Menos de 65 puntos de función
Mediano	Entre 65 y 110 puntos de función
Grande	Más de 110 puntos de función

Tabla 10. Tipos de Proyectos de Software

Tipo de Proyecto	ID de Caso	N° de Tareas	Max. Hab / Tarea	N° de Desarr.	Max. Hab / Desarr.
Pequeño	P1	24	2	2	2
	P2	24	2	5	2
	P3	24	2	2	10
	P4	24	2	5	10
	P5	24	10	2	10
	P6	24	10	5	10
Mediano	M1	54	2	2	2
	M2	54	2	5	2
	M3	54	2	2	10
	M4	54	2	5	10
	M5	54	10	2	10
	M6	54	10	5	10
Grande	G1	131	2	2	2
	G2	131	2	5	2
	G3	131	2	2	10
	G4	131	2	5	10
	G5	131	10	2	10
	G6	131	10	5	10

Tabla 11. Definición de Casos de Prueba

ID	Metaheurística
SA	Recocido Simulado
VNS	Búsqueda con vecindad variable
AG5	Algoritmo Genético (Probabilidad de mutación = 5%)
AG10	Algoritmo Genético (Probabilidad de mutación = 10%)
AG15	Algoritmo Genético (Probabilidad de mutación = 15%)

Tabla 12. Identificación de las Metaheurísticas

5.2 Evaluación de los resultados

La obtención de los resultados se llevó a cabo ejecutando cada metaheurística (identificadas en la Tabla 12) para cada uno de los casos de prueba 10 veces, tanto para el caso de reducción de tiempo (asignando expertos o no) como de costo, indicados como variantes de la función objetivo en la representación del problema, conformando así una muestra pequeña ($n \leq 30$) sobre la cual realizar las pruebas respectivas. Es importante destacar que dado que las metaheurísticas son de recorrido aleatorio, se requiere del conjunto particular de ejecuciones para realizar una estimación estadística a través del análisis de las medias de las soluciones obtenidas en cada ejecución, lo cual permite indicar en primera instancia qué algoritmo se comporta en general mucho mejor que otro. Los valores correspondientes a las medias para cada instancia de cada tipo de proyecto obtenidos a partir de dichas ejecuciones se muestran en las tablas 13-15.

Dado que se desconoce la significación de los datos obtenidos es fundamental la realización de pruebas no paramétricas, entendidas como aquellas que no presuponen una distribución de probabilidad para los datos (García, Molina, Herrera, & Lozano, 2009). Dichas pruebas permiten determinar sobre un conjunto de datos la posible relación que existen entre sus muestras, o en contraparte su independencia, comprobar si existen diferencias significativas, etc.

En el Anexo C, se indican las desviaciones estándar de los resultados obtenidos para todos los casos de prueba.

Caso de Prueba	Metaheurística				
	SA	**VNS**	**AG5**	**AG10**	**AG15**
P1	5843.55	**4491.85**	4709.4	4759.45	4719.35
P2	4985.27	**3424.52**	3781.38	3894.364	3828.99
P3	5937.75	**4498.23**	4710.2	4743.55	4706.9
P4	5166.22	**3368.52**	3853.82	3911.31	3869.71
P5	6322	**5647**	5704.5	5742	5714.5
P6	4729.81	3695.78	3831.81	**3641.55**	3714.06
M1	10027.6	**8755.9**	9151.8	9078.1	9062.65
M2	8064.98	**6514.52**	7207.87	6942.59	7048.99
M3	9981.85	**8718.05**	9259.3	9188.85	9103.1
M4	8472.18	**6608.76**	7074.16	7675.16	6866.35
M5	10304.6	**9418.35**	9730.5	9703	9745.05
M6	7640.92	**6223.44**	6809.94	6635.83	6770.82
G1	12562	**11557**	**11557**	11910.1	11902.75
G2	9823.2	**8276.57**	9136.42	9154.92	9199.72
G3	12532.2	**11573.5**	11977.5	11901.7	11904.75
G4	10285.2	**8266.07**	9143.52	9138.35	9080.47
G5	12655.5	**11658.5**	12094.7	12082.5	11951.7
G6	9323.67	**8047.92**	8604.92	8699.92	8728.92

Tabla 13. Medias de los resultados obtenidos para la reducción de costos a partir de metaheurísticas simples

Caso de Prueba	Metaheurística				
	SA	**VNS**	**AG5**	**AG10**	**AG15**
P1	60.201	54.5	**43.017**	55.135	55.035
P2	49.634	**44.567**	45.517	45.467	45.8
P3	61.136	**54.517**	55.118	55.268	55.134
P4	50.617	**44.55**	45.601	45.684	45.067
P5	62.469	**55.551**	56.402	56.484	55.851
P6	50.317	**41.317**	43.017	42.934	42.884
M1	93.58	**72.869**	78.58	78.336	77.963
M2	68.186	**48.377**	51.415	52.51	51.977
M3	92.856	**75.561**	78.82	77.496	78.866
M4	63.518	**52.269**	53.452	53.203	53.485
M5	100.18	**82.626**	86.394	87.402	85.087
M6	67.494	**47.526**	52.493	52.743	52.086
G1	116.81	**94.125**	102.285	102.61	103.355
G2	68.245	**53.945**	57.135	56.965	57.895
G3	114.93	114.93	103.08	102.565	**101.435**
G4	67.33	**54.71**	57.375	56.74	58.12
G5	116.25	**95.691**	103.265	102.69	103.285
G6	63.23	**44.8**	50.97	50.375	50.775

Tabla 14. Medias de los resultados obtenidos para la reducción de tiempo a partir de metaheurísticas simples

Caso de Prueba	Metaheurística				
	SA	**VNS**	**AG5**	**AG10**	**AG15**
P1	61.306	55.2	**43.13**	55.205	56.017
P2	50.015	**44.630**	44.978	46.013	45.312
P3	61.756	55.023	55.012	56.301	**54.605**
P4	52.318	**44.914**	46.702	46.621	45.451
P5	62.9	**55.97**	56.301	57.861	56.317
P6	50.561	**41.869**	44.521	43.238	43.628
M1	94.856	**72.01**	77.63	77.856	77.521
M2	69.85	**50.12**	52.5	51.68	52.019
M3	93.652	**73.25**	79.30	77.6	78.941
M4	64.101	**50.854**	54.5	53.814	55.632
M5	101.01	82.512	86.849	87.68	**83.985**
M6	67.05	**48.62**	52.87	53.92	52.478
G1	116.8	**94.478**	100.56	101.74	103.651
G2	65.902	**53.945**	55.15	54.95	55.95
G3	115.01	113.02	**101.05**	105.08	103.5
G4	67.56	**55.01**	57.89	57.85	59.62
G5	117.56	**97.45**	101.54	105.56	104.899
G6	65.89	**46.8**	55.43	50.124	50.536

Tabla 15. Medias de los resultados obtenidos para la reducción de costos a partir de metaheurísticas simples asignando desarrolladores expertos

Entre las pruebas no paramétricas, se encuentra la prueba de Friedman (García, Molina, Herrera, & Lozano, 2009), la cual es equivalente a una prueba ANOVA[2] para dos factores y permite determinar si en un conjunto de pruebas existen

[2] El análisis de la varianza (denominado ANOVA por su nombre en inglés ---ANalysis Of VAriance---) es un modelo en el cual la varianza está representada por una variable cuya variación

diferencias significativas entre sí, y si las mismas se encuentran interrelacionadas. El cálculo se basa en rango, que luego es comparado con una distribución Chi^2 con grados de libertad k-1. A continuación se describe el procedimiento realizado para el análisis de datos.

- **Evaluación de la existencia de diferencias significativas.** La existencia de diferencias significativas entre los datos fue determinada a partir del Test de Friedman, bajo el siguiente escenario:

  H_0: No hay diferencia entre los resultados obtenidos
  H_1: Si hay diferencia entre los resultados obtenidos
  K: grados de libertad
  Prob Chi^2: Valor critico de Chi^2 con k-1 grados de libertad
  Valor de Friedman: estadístico calculado del análisis de varianza por rangos de Friedman
  Si el Valor de Friedman es mayor que el punto crítico de la Prob Chi^2, entonces se rechaza H_0, caso contrario se rechaza H_1.

Los resultados del Test de Friedman bajo el escenario indicado se muestran en las Tablas 16-18, en las que se puede observar que dado que el valor obtenido de Friedman es mucho mayor en todos los casos que la probabilidad de Chi^2, hay pruebas estadísticas suficientes para rechazar la hipótesis nula y concluir que efectivamente existen diferencias significativas entre los resultados obtenidos por cada metaheurística para todos los tipos de proyectos evaluados, indicando de esta

depende de diferentes fuentes (otras variables) y permite además, la realización de pruebas de significación estadística.

manera que algunas de las estrategias implementadas resultaron más efectivas que otras para el problema evaluado.

Tipo de Proyecto	Valor de Friedman	Prob Chi^2	K – 1	Punto Crítico	Se rechaza
Pequeño	19.36	0.001	4	9.4877	H_0
Mediano	17.12	0.002	4	9.4877	H_0
Grande	15.32	0.004	4	9.4877	H_0

Tabla 16. Resultados de la prueba de Friedman para reducción de costos aplicando metaheurísticas simples

Tipo de Proyecto	Valor de Friedman	Prob Chi^2	K – 1	Punto Crítico	Se rechaza
Pequeño	16	0.003	4	9.4877	H_0
Mediano	16	0.003	4	9.4877	H_0
Grande	12.28	0.02	4	9.4877	H_0

Tabla 17. Resultados de la prueba de Friedman para reducción de tiempo aplicando metaheurísticas simples

Tipo de Proyecto	Valor de Friedman	Prob Chi^2	K – 1	Punto Crítico	Se rechaza
Pequeño	13.83	0.01	4	9.4877	H_0
Mediano	19.73	0.001	4	9.4877	H_0
Grande	13.83	0.01	4	9.4877	H_0

Tabla 18. Resultados de la prueba de Friedman para reducción de tiempo aplicando metaheurísticas simples asignando desarrolladores expertos

- **Evaluación de diferencias significativas**: una vez comprobada la existencia de diferencias significativas entre los algoritmos, resta determinar cuáles son. Para ello, se llevó a cabo el Test de Holm (García, Molina, Herrera, & Lozano, 2009), a partir del cual se comienza ordenando por rangos los valores asociados con el estadístico de la prueba (p) y se selecciona el valor más pequeño como algoritmo de control. De esta manera, si el valor estadístico es menor que el valor de comparación (tenor) se concluye que existen diferencias significativas. Los resultados del Test de Holm para cada tipo de proyecto y para cada función objetivo se indican en las tablas 19 – 27.

Algoritmo	Tenor	Valor	¿Hay Dif.?
AG5	0.05	0.11506967022	No
AG15	0.025	0.03593031911	No
AG10	0.016666	0.00134989803	Si
SA	0.0125	0.00003167124	Si

Tabla 19. Resultados de la prueba de Holm para reducción de costos en proyectos pequeños aplicando metaheurísticas simples. Algoritmo de Control: VNS

Algoritmo	Tenor	Valor	¿Hay Dif.?
AG15	0.05	0.05479929169	No
AG10	0.025	0.03593031911	No
AG5	0.016666	0.00466118802	Si
SA	0.0125	0.00003167124	Si

Tabla 20. Resultados de la prueba de Holm para reducción de costos en proyectos medianos aplicando metaheurísticas simples. Algoritmo de Control: VNS

Algoritmo	Tenor	Valor	¿Hay Dif.?
AG15	0.05	0.05456546275	No
AG10	0.025	0.02871655981	No
AG5	0.016666	0.02275013194	No
SA	0.0125	0.00004809634	Si

Tabla 21. Resultados de la prueba de Holm para reducción de costos en proyectos grandes aplicando metaheurísticas simples. Algoritmo de Control: VNS

Algoritmo	Tenor	Valor	¿Hay Dif.?
AG5	0.05	0.11506967022	No
AG15	0.025	0.05479929169	No
AG10	0.016666	0.00819753592	Si
SA	0.0125	0.00007234804	Si

Tabla 22. Resultados de la prueba de Holm para reducción de tiempo en proyectos pequeños aplicando metaheurísticas simples. Algoritmo de Control: VNS

Algoritmo	Tenor	Valor	¿Hay Dif.?
AG5	0.05	0.11506967022	No
AG15	0.025	0.05479929169	No
AG10	0.016666	0.00819753592	Si
SA	0.0125	0.00007234804	Si

Tabla 23. Resultados de la prueba de Holm para reducción de tiempo en proyectos medianos aplicando metaheurísticas simples. Algoritmo de Control: VNS

Algoritmo	Tenor	Valor	¿Hay Dif.?
AG10	0.05	0.30853753872	No
AG5	0.025	0.13566606094	No
AG15	0.016666	0.04456546275	No
SA	0.0125	0.00068713793	Si

Tabla 24. Resultados de la prueba de Holm para reducción de tiempo en proyectos grandes aplicando metaheurísticas simples. Algoritmo de Control: VNS

Algoritmo	Tenor	Valor	¿Hay Dif.?
AG5	0.05	0.15524721715	No
AG15	0.025	0.11836178531	No
AG10	0.016666	0.031489525607	No
SA	0.0125	0.00019287337	Si

Tabla 25. Resultados de la prueba de Holm para reducción de tiempo asignando desarrolladores Expertos en proyectos pequeños aplicando metaheurísticas simples - Algoritmo de Control: VNS

Algoritmo	Tenor	Valor	¿Hay Dif.?
AG15	0.05	0.03394457743	Si
AG10	0.025	0.01422986845	Si
AG5	0.016666	0.00529356866	Si
SA	0.0125	0.00000588566	Si

Tabla 26. Resultados de la prueba de Holm para reducción de tiempo asignando desarrolladores expertos en proyectos medianos aplicando metaheurísticas simples - Algoritmo de Control: VNS

Algoritmo	Tenor	Valor	¿Hay Dif.?
AG5	0.05	0.15524721715	No
AG10	0.025	0.11836178531	No
AG15	0.016666	0.03148952560	No
SA	0.0125	0.00019287337	Si

Tabla 27. Resultados de la prueba de Holm para reducción de tiempo asignando desarrolladores expertos en proyectos grandes aplicando metaheurísticas simples - Algoritmo de Control: VNS

De acuerdo con los resultados obtenidos y su correspondiente análisis fue posible determinar que tanto el VNS como el algoritmo genético (con sus distintas probabilidades de mutación) arrojaron mejores resultados para todos los escenarios de prueba que el SA. Para ilustrar podemos tomar como ejemplo la instancia 1 de los proyectos medianos (M1), donde el GP logró crear una planificación centrada en la reducción de tiempo con una duración de aproximadamente 80 horas, valor que en contraste con los resultados reflejados en la Tabla 14 es mejor que el obtenido por el SA pero inferior a los obtenidos con el VNS y las distintas versiones del AG. Este resultado es similar para el resto de las instancias.

Análisis de Sensibilidad del Algoritmo Genético

Para efectos de observar la incidencia de los parámetros propios del algoritmo genético en la generación de las planificaciones, a continuación se exponen los resultados de un análisis de sensibilidad llevado a cabo variando la probabilidad de mutación y número de generaciones del AG para reducción de tiempo, consolidando los casos de prueba indicados en la Tabla 28, para cada uno de los cuales se mantuvo

el mismo número de evaluaciones. Las medias de los resultados de dichas pruebas se indican en la Tabla 29, cuyo análisis es representado en el diagrama de cajas y bigotes de la Figura 21. En particular se observar que los mejores resultados promedios y con menor desviación estándar, se obtuvieron a partir del 15% de probabilidad de mutación y 40 generaciones. Para determinar el grado de significación se realizó el Test Holm ---como se observa en la Tabla 30--- lo que nos indica que no hay suficientes indicios para aseverar que es la mejor opción de configuración de los parámetros del GA.

ID	Probabilidad de Mutación	Número de Generaciones
GA_A	5%	10
GA_B		20
GA_C		40
GA_D	10%	10
GA_E		20
GA_F		40
GA_G	15%	10
GA_H		20
GA_I		40

Tabla 28. Identificación de las versiones del Algoritmo Genético para el análisis de sensibilidad (Reducción de Tiempo)

Caso de Prueba	Algoritmo Genético								
	GA_A	**GA_B**	**GA_C**	**GA_D**	**GA_E**	**GA_F**	**GA_G**	**GA_H**	**GA_I**
P1	56.23	57.51	**56.02**	56.07	56.13	57.79	56.65	57.49	56.07
P2	48.88	49.80	50.28	49.88	**48.60**	49.62	48.73	49.34	50.30
P3	63.57	62.82	62.57	62.51	62.63	62.74	62.36	**62.33**	63.58
P4	51.97	52.68	51.04	51.13	51.36	51.38	51.78	51.78	**50.25**
P5	56.88	**56.72**	57.54	57.24	56.83	56.96	56.95	57.13	56.75
P6	44.52	45.44	44.97	45.22	44.77	**44.37**	45.04	44.49	44.40
M1	80.49	80.35	78.28	81.35	77.99	77.54	**77.00**	80.13	79.48
M2	60.25	59.15	61.06	60.61	60.01	**58.43**	60.14	59.46	58.94
M3	79.14	77.94	79.99	**77.87**	80.02	80.74	79.64	78.26	78.95
M4	69.40	68.23	67.47	72.15	64.86	69.56	68.12	**64.76**	67.41
M5	85.99	85.77	86.33	85.37	85.50	86.17	**84.36**	85.98	85.47
M6	69.61	69.83	68.26	**67.42**	69.91	73.09	70.73	67.55	72.88
G1	101.67	102.84	104.58	102.18	**100.94**	101.02	102.39	102.49	100.95
G2	62.97	62.69	61.26	62.51	**59.54**	63.22	63.66	64.25	59.76
G3	**101.39**	102.02	102.13	101.93	101.64	102.76	102.33	103.61	101.59
G4	**58.08**	61.46	61.26	61.15	62.22	61.42	62.80	58.92	63.06
G5	104.34	104.33	104.77	103.42	103.52	105.18	103.33	**103.23**	102.39
G6	51.81	**50.40**	51.41	51.39	51.42	50.75	51.90	52.82	50.61

Tabla 29. Medias de los resultados obtenidos para reducción de tiempo con distintas versiones del algoritmo genético (Análisis de Sensibilidad)

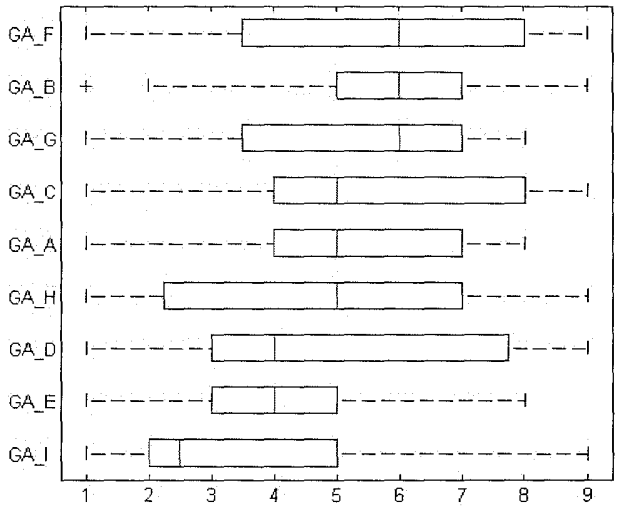


Figura 21. Diagrama de Cajas y Bigotes: Versiones del AG en el análisis de sensibilidad

Algoritmo	Tenor	Valor	¿Hay Dif.?
GA_E	0.05	0.3956	No
GA_D	0.025	0.1577	No
GA_H	0.016	0.118	No
GA_G	0.0125	0.0617	No
GA_A	0.01	0.0515	No
GA_B	0.0083	0.0354	No
GA_C	0.0071	0.0271	No
GA_F	0.00625	0.0177	No

Tabla 30. Resultados de la prueba de Holm para reducción de tiempo con distintas versiones del algoritmo genético - Algoritmo de Control: GA_I

5.3 Implementación de un Algoritmo Memético

Con base en dichos resultados, y considerando las ventajas de la hibridación de metaheurísticas expuestas en diversas investigaciones (Martínez, 2011), se formuló un algoritmo memético clásico tal como se indica en la Figura 22. Este algoritmo se basa en la estructura de un algoritmo poblacional, que en este caso es el algoritmo genético, e incluye bajo cierta probabilidad de ocurrencia un mecanismo de una metaheurística local o de trayectoria, que en este caso es el VNS el cual se desarrolla de acuerdo al pseudocódigo especificado en la Figura 5.

Para las pruebas correspondientes al algoritmo memético implementado, se definieron los escenarios de prueba indicados en la Tabla 31, cada uno de los cuales tiene una probabilidad de mutación propia del algoritmo genético así como también una probabilidad de ejecución del VNS para efectos de realizar las comparaciones correspondientes.

Los valores correspondientes a las medias para cada escenario de prueba, obtenidos a partir de las ejecuciones del algoritmo memético las cuales se realizaron de la misma forma que las metaheurísticas simples, se indican en las Tablas 32 – 34, y su correspondiente interpretación a través de las pruebas de Friedman y Holm en las Tablas 35-46. La representación gráfica de las diferencias entre las metaheurísticas simples y las diferentes versiones del algoritmo memético implementadas se indica en la Tabla 47.

```
1  P; // conjunto de soluciones de tamaño N, i.e., {S_1, S_2, ..., S_N}
2  FUNCTION MemeticAlgorithm
3  P ← InitPopulation();
4  EvaluateFitness(P);
5  S_φ ← GetBestSolution(P);
6  while stopping criterion is not reached do
7      while i ∈ N/2 do
8          Q ←SelectParents(P); // selecciona 2 soluciones candidatas
9          if Rand[0,1) < p_crossover then
10             Q ←Crossover(Q);
11         end if
12         for i ∈ Q do
13             if Rand[0,1) < p_mutation then
14                 Q[i] ←Mutation(Q[j])
15             end if
16             if Rand[0,1) < p_localsearch then
17                 Q[i] ←LocalImprovement(Q[j]); // aplicación de un
18                                               // procedimiento de mejora
19             end if
20         end for
21         P ← ReplaceWorstIndividuals(P, Q);
22     end while
23     S_φ ← GetBestSolution(P, S_φ);
24 end while
25 return S_φ
```

Figura 22. Pseudocódigo del Algoritmo Memético

ID	Algoritmo Memético	
	Probabilidad de mutación	**Probabilidad de Búsqueda Local**
MA_M5_VNS15	5%	15%
MA_M5_VNS30	5%	30%
MA_M10_VNS15	10%	15%
MA_M10_VNS30	10%	30%
MA_M15_VNS15	15%	15%
MA_M15_VNS30	15%	30%

Tabla 31. Identificación del Algoritmo Memético

Caso de Prueba	Algoritmo Memético					
	MA_M5_VNS15	**MA_M5_VNS30**	**MA_M10_VNS15**	**MA_M10_VNS30**	**MA_M15_VNS15**	**MA_M15_VNS30**
P1	4685.3	4676.1	4652.75	**4630.25**	4646.9	4699.4
P2	3800.02	3823.475	3783.88	3896.93	**3763.645**	3796.325
P3	4663.6	**4640.2**	4691.1	4711.05	4744.4	4671.1
P4	3854.88	3885.525	**3698.89**	3853.64	3863.33	3778.29
P5	**5672**	5687	5679.5	5692	5727	5682
P6	**3657.55**	3708.52	3724.04	3758.8	3798.32	3776.8
M1	9036.8	9048.45	9056.85	9131.45	9030.95	**9002.2**
M2	**6898.015**	7049.9	7023.995	7082.63	6954.05	6979.675
M3	9146.4	9144.7	**9032.25**	9176.4	9066	9159.85
M4	7021.5	7027.306	7052.73	7075.05	**6837.53**	7013.065
M5	9616.75	9643	9637.2	**9580.1**	9718.85	9625.9
M6	6810.645	6803.305	**6714.58**	6805.485	6843.785	6785.815
G1	11927.75	11925	**11893.75**	11929.25	11942.25	11954.75
G2	**9047.125**	9147.05	9120.675	9231.34	9175.225	9326.9
G3	11919.75	11921.25	**11874.5**	11885.75	11917.75	11915.5
G4	**9064.525**	9246.3	9172.775	9122.25	9167.5	9197.4
G5	12045	12068.25	11947	**11006.75**	11966.75	12049.5
G6	**8609.125**	8730.4	8665.55	8766.775	8645.55	8640.85

Tabla 32. Medias de los resultados obtenidos para la reducción de costos a partir de la aplicación del algoritmo memético

Caso de Prueba	Algoritmo Memético					
	MA_M5_VNS15	**MA_M5_VNS30**	**MA_M10_VNS15**	**MA_M10_VNS30**	**MA_M15_VNS15**	**MA_M15_VNS30**
P1	55.268	54.885	**54.884**	54.951	55.201	55.118
P2	**45.118**	45.467	45.267	45.25	45.267	45.284
P3	55.301	**54.918**	55.251	55.068	55.051	55.153
P4	45.2	45.2	**45**	45.05	45.267	45.184
P5	55.918	55.868	55.834	56.052	**55.684**	55.818
P6	**42.2**	42.401	42.801	42.601	42.317	42.684
M1	77.33	79.329	77.862	79.154	77.472	**77.154**
M2	52.243	**51.402**	51.587	52.386	52.427	52.794
M3	78.818	77.645	**77.104**	79.005	78.063	78.67
M4	53.428	52.403	**51.803**	52.042	52.885	51.985
M5	**84.561**	85.877	85.279	86.012	85.496	85.295
M6	52.352	51.318	52.318	51.32	**50.925**	51.601
G1	102.415	104.825	102.805	103.47	**101.975**	103.155
G2	57.45	57.75	57.54	57.325	**58.075**	57.615
G3	102.635	102.86	103.62	**101.885**	103.115	102.885
G4	57.515	**57.21**	57.505	57.515	58.1	57.385
G5	**101.715**	103.78	103.15	103.28	102.04	103.01
G6	51.04	51.375	**49.835**	51.225	52.165	50.575

Tabla 33. Medias de los resultados obtenidos para la reducción de tiempo a partir de la aplicación del algoritmo memético

Caso de Prueba	Algoritmo Memético					
	MA_M5_VNS15	MA_M5_VNS30	MA_M10_VNS15	MA_M10_VNS30	MA_M15_VNS15	MA_M15_VNS30
P1	56.352	55.651	55.852	55.902	**55.319**	55.834
P2	**45.767**	45.968	45.967	45.834	46.717	45.934
P3	55.868	55.818	56.169	**55.585**	55.652	55.568
P4	46.068	46.634	46.867	46.601	46.584	**45.584**
P5	61.602	62.503	62.703	61.303	**61.17**	61.553
P6	**45.651**	47.519	46.868	45.685	46.285	46.601
M1	77.748	78.462	**77.205**	78.63	77.206	78.222
M2	53.485	55.527	**53.194**	55.262	54.852	55.052
M3	78.171	78.938	**77.588**	78.996	78.156	77.905
M4	57.541	53.36	**53.211**	55.528	52.336	53.862
M5	85.662	87.045	85.594	**85.27**	85.913	85.602
M6	53.702	54.686	**53.177**	55.15	55.068	52.61
G1	102.025	102.83	100.565	101.475	**102.2**	101.485
G2	58.145	57.96	58.99	57.83	**57.26**	57.465
G3	102.38	102.77	102.685	101.96	102.495	**101.435**
G4	56.69	57.875	57.635	59.2	**56.355**	57.665
G5	105.91	104.61	104.675	104.68	103.92	**103.425**
G6	52.3	53.905	54.17	**52.02**	53.485	54.24

Tabla 34. Medias de los resultados obtenidos para la reducción de tiempo asignando desarrolladores expertos a partir de la aplicación del algoritmo memético

Tipo de Proyecto	Valor de Friedman	Prob Chi^2	K – 1	Punto Crítico	Se rechaza
Pequeño	34.58	0.0001	10	18.3070	H_0
Mediano	34.87	0.0001	10	18.3070	H_0
Grande	29.46	0.0010	10	18.3070	H_0

Tabla 35. Resultados de la prueba de Friedman para reducción de costos aplicando el algoritmo memético

Tipo de Proyecto	Valor de Friedman	Prob Chi^2	K – 1	Punto Crítico	Se rechaza
Pequeño	29.18	0.0012	10	18.3070	H_0
Mediano	27.16	0.0025	10	18.3070	H_0
Grande	22.31	0.0136	10	18.3070	H_0

Tabla 36. Resultados de la prueba de Friedman para reducción de tiempo aplicando el algoritmo memético

Tipo de Proyecto	Valor de Friedman	Prob Chi^2	K – 1	Punto Crítico	Se rechaza
Pequeño	31.05	0.001	10	18.3070	H_0
Mediano	30.95	0.0001	10	18.3070	H_0
Grande	24.44	0.007	10	18.3070	H_0

Tabla 37. Resultados de la prueba de Friedman para reducción de tiempo asignando desarrolladores expertos aplicando el algoritmo memético

Algoritmo	Tenor	Valor	¿Hay Dif.?
MA_M10_VNS15	0.0500	0.1076	No
MA_M5_VNS15	0.0250	0.0431	No
MA_M15_VNS30	0.0167	0.0431	No
MA_M5_VNS30	0.0125	0.0142	No
AG5	0.0100	0.0086	Si
MA_M10_VNS30	0.0083	0.0086	No
MA_M15_VNS15	0.0071	0.0066	Si
AG15	0.0063	0.0006	Si
AG10	0.0056	0.0000	Si
SA	0.0050	0.0000	Si

Tabla 38. Resultados de la prueba de Holm para reducción de costos en proyectos pequeños aplicando el algoritmo memético. Algoritmo de Control: VNS

Algoritmo	Tenor	Valor	¿Hay Dif.?
MA_M5_VNS15	0.0500	0.0763	No
MA_M15_VNS15	0.0250	0.0763	No
MA_M15_VNS30	0.0167	0.0525	No
MA_M10_VNS15	0.0125	0.0226	No
MA_M5_VNS30	0.0100	0.0086	Si
AG15	0.0083	0.0066	Si
AG10	0.0071	0.0011	Si
MA_M10_VNS30	0.0063	0.0011	Si
AG5	0.0056	0.0000	Si
SA	0.0050	0.0000	Si

Tabla 39. Resultados de la prueba de Holm para reducción de costos en proyectos medianos aplicando el algoritmo memético. Algoritmo de Control: VNS

Algoritmo	Tenor	Valor	¿Hay Dif.?
MA_M10_VNS15	0.0500	0.1167	No
AG15	0.0250	0.0476	No
MA_M5_VNS15	0.0167	0.0389	No
MA_M10_VNS30	0.0125	0.0389	No
AG10	0.0100	0.0160	No
AG5	0.0083	0.0086	No
MA_M15_VNS15	0.0071	0.0033	Si
MA_M5_VNS30	0.0063	0.0013	Si
MA_M15_VNS30	0.0056	0.0004	Si
SA	0.0050	0.0000	Si

Tabla 40. Resultados de la prueba de Holm para reducción de costos en proyectos grandes aplicando el algoritmo memético. Algoritmo de Control: VNS

Algoritmo	Tenor	Valor	¿Hay Dif.?
MA_M10_VNS15	0.0500	0.0697	No
MA_M10_VNS30	0.0250	0.0525	No
MA_M5_VNS30	0.0167	0.0283	No
MA_M15_VNS15	0.0125	0.0253	No
MA_M15_VNS30	0.0100	0.0180	No
AG15	0.0083	0.0086	No
AG5	0.0071	0.0050	Si
MA_M5_VNS15	0.0063	0.0025	Si
AG10	0.0056	0.0001	Si
SA	0.0050	0.0000	Si

Tabla 41. Resultados de la prueba de Holm para reducción de tiempo en proyectos pequeños aplicando el algoritmo memético. Algoritmo de Control: VNS

Algoritmo	Tenor	Valor	¿Hay Dif.?
MA_M10_VNS15	0.0500	0.2228	No
MA_M5_VNS15	0.0250	0.0525	No
MA_M15_VNS30	0.0167	0.0431	No
MA_M5_VNS30	0.0125	0.0283	No
MA_M15_VNS15	0.0100	0.0226	No
AG15	0.0083	0.0086	No
AG10	0.0071	0.0038	Si
AG5	0.0063	0.0028	Si
MA_M10_VNS30	0.0056	0.0028	Si
SA	0.0050	0.0000	Si

Tabla 42. Resultados de la prueba de Holm para reducción de tiempo en proyectos medianos aplicando el algoritmo memético. Algoritmo de Control: VNS

Algoritmo	Tenor	Valor	¿Hay Dif.?
AG10	0.0500	0.4431	No
MA_M5_VNS15	0.0250	0.2228	No
AG5	0.0167	0.01825	No
MA_M15_VNS30	0.0125	0.0697	No
MA_M10_VNS30	0.0100	0.0636	No
MA_M15_VNS15	0.0083	0.0476	No
MA_M10_VNS15	0.0071	0.0389	No
MA_M5_VNS30	0.0063	0.0202	No
AG15	0.0056	0.0160	No
SA	0.0050	0.0001	Si

Tabla 43. Resultados de la prueba de Holm para reducción de tiempo en proyectos grandes aplicando el algoritmo memético. Algoritmo de Control: VNS

Algoritmo	Tenor	Valor	¿Hay Dif.?
AG5	0.0500	0.2228	No
AG15	0.0250	0.1702	No
MA_M15_VNS30	0.0167	0.0431	No
MA_M10_VNS30	0.0125	0.0180	No
MA_M15_VNS15	0.0100	0.0180	No
AG10	0.0083	0.0086	No
MA_M5_VNS15	0.0071	0.0066	Si
MA_M5_VNS30	0.0063	0.0028	Si
MA_M10_VNS15	0.0056	0.0004	Si
SA	0.0050	0.0000	Si

Tabla 44. Resultados de la prueba de Holm para reducción de tiempo en proyectos pequeños con asignación de desarrolladores expertos aplicando el algoritmo memético. Algoritmo de Control: VNS

Algoritmo	Tenor	Valor	¿Hay Dif.?
MA_M10_VNS15	0.0500	0.1471	No
MA_M15_VNS15	0.0250	0.0350	No
AG15	0.0167	0.0226	No
AG10	0.0125	0.0180	No
MA_M15_VNS30	0.0100	0.0066	Si
AG5	0.0083	0.0028	Si
MA_M5_VNS15	0.0071	0.0028	Si
MA_M5_VNS30	0.0063	0.0006	Si
MA_M10_VNS30	0.0056	0.0006	Si
SA	0.0050	0.0000	Si

Tabla 45. Resultados de la prueba de Holm para reducción de tiempo en proyectos medianos con asignación de desarrolladores expertos aplicando el algoritmo memético. Algoritmo de Control: VNS

Algoritmo	Tenor	Valor	¿Hay Dif.?
AG5	0.0500	0.4244	No
MA_M15_VNS30	0.0250	0.2523	No
MA_M15_VNS15	0.0167	0.1702	No
MA_M10_VNS15	0.0125	0.0763	No
MA_M10_VNS30	0.0100	0.0636	No
AG10	0.0083	0.0431	No
MA_M5_VNS15	0.0071	0.0350	No
MA_M5_VNS30	0.0063	0.0180	No
AG15	0.0056	0.0066	No
SA	0.0050	0.0000	Si

Tabla 46. Resultados de la prueba de Holm para reducción de tiempo en proyectos grandes con asignación de desarrolladores expertos aplicando el algoritmo memético. Algoritmo de Control: VNS

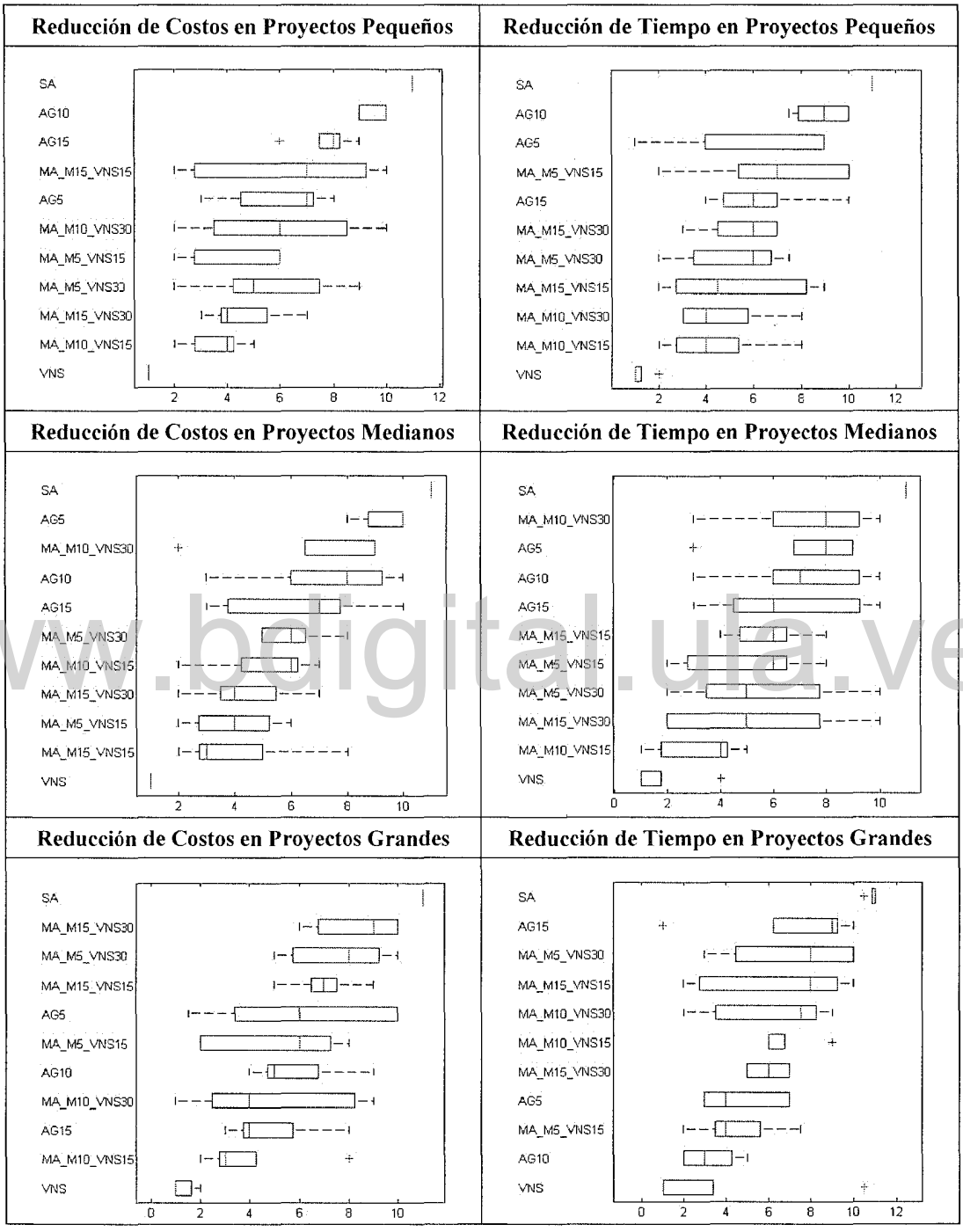


Tabla 47. Diagramas de Cajas y Bigotes de los resultados del algoritmo memético para reducción de costos y tiempo

Análisis de sensibilidad del algoritmo memético

En el mismo orden de ideas para efectos de tomar en cuenta los diversos parámetros que juegan un papel importante para los resultados obtenidos en el AM, a continuación se detallan algunas pruebas realizadas con 10% de probabilidad de mutación (siendo que fue el valor que arrojó los mejores resultados para todos los casos en general) y distintas probabilidades de búsqueda local tal como se indica en la Tabla 48, con el fin de verificar la incidencia del mecanismo de lamarkismo parcial dentro del AM. Las medias de los resultados de dichas pruebas, las cuales se realizaron con el mismo número de evaluaciones, se indican en la Tabla 49 los cuales son representados gráficamente en el diagrama de cajas y bigotes de la Figura 23, en la que se puede observar que con la configuración de 15% de probabilidad de aplicación del método de búsqueda local VNS en el memético y con un probabilidad de mutación de 10% se obtuvo en 7 de los 18 casos los mejores resultados para el problema abordado.

ID	Probabilidad de Mutación	Probabilidad de Búsqueda Local
MA_A	10%	5%
MA_B		10%
MA_C		15%
MA_D		20%
MA_E		25%
MA_F		30%

Tabla 48. Identificación de las versiones del Algoritmo Memético para el análisis de sensibilidad (Reducción de Tiempo)

Caso de Prueba	Algoritmo Memético					
	MA_A	**MA_B**	**MA_C**	**MA_D**	**MA_E**	**MA_F**
P1	57.04	**55.96**	57.09	57.43	57.07	56.52
P2	50.33	49.03	**48.94**	51.02	49.80	50.12
P3	61.72	63.06	**61.04**	62.50	61.64	61.90
P4	**49.80**	50.18	52.09	50.86	52.36	50.01
P5	56.80	**56.88**	57.06	56.93	57.30	56.98
P6	45.32	45.43	44.78	46.02	**44.57**	45.25
M1	80.51	**79.80**	80.73	81.90	81.36	81.41
M2	58.20	60.13	**56.86**	59.29	58.34	62.39
M3	81.41	80.77	80.67	80.57	80.50	**80.30**
M4	66.46	65.66	66.17	**64.56**	66.73	68.40
M5	87.13	86.99	**86.25**	86.68	86.75	86.41
M6	66.96	**65.08**	66.15	66.35	65.56	67.10
G1	105.09	105.21	**103.69**	104.73	104.88	104.97
G2	65.18	63.74	64.19	63.98	62.50	**62.08**
G3	104.05	104.18	**104.02**	104.92	104.28	104.55
G4	62.61	61.23	61.79	**59.52**	63.31	61.29
G5	105.83	106.17	**105.45**	105.83	105.89	105.71
G6	54.71	54.22	54.94	**53.22**	53.58	56.24

Tabla 49. Medias de los resultados obtenidos para reducción de tiempo con distintas versiones del algoritmo memético (Análisis de Sensibilidad)

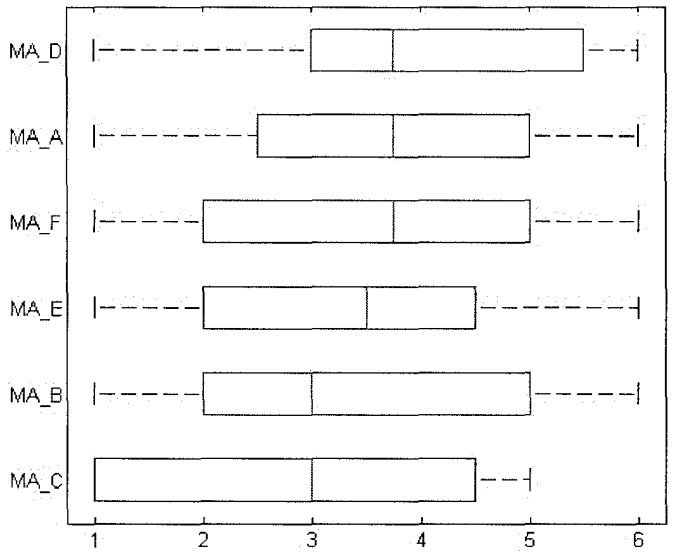


Figura 23. Diagrama de Cajas y Bigotes: Versiones del AM en el análisis de sensibilidad

5.4 Comparación General

A partir de los resultados obtenidos mediante la ejecución de las pruebas de las metaheurísticas simples y el algoritmo memético respectivamente, es posible afirmar el VNS es el algoritmo que arrojó los mejores resultados para todos los casos de prueba en lo concerniente a reducción de costo y reducción de tiempo, con o sin asignación de desarrolladores expertos.

Sin embargo, para efectos de realizar una comparación más general en la que se visualice con mayor claridad dicha afirmación, a continuación se presenta un resumen de los resultados obtenidos para reducción de tiempo a partir de todas las

metaheurísticas implementadas, especificando las versiones del algoritmo genético y memético que en general arrojaron mejores resultados para todos los casos de prueba.

En la Tabla 50 se indican las medias de los valores obtenidos por las estrategias incluidas en este resumen y en las tablas 51 – 54 se indican los resultados correspondientes a las pruebas de Friedman y Holm de este subconjunto de ejecuciones.

Adicionalmente, los diagramas de cajas y bigotes indicados en la Tabla 55 corresponden a la representación gráfica de los resultados obtenidos en esta comparación general, en los que se puede observar: la poca dispersión de los resultados del VNS y la simetría de los resultados del algoritmo memético, así como también las diferencias significativas que existen con respecto a los resultados obtenidos a partir del SA.

Caso de Prueba	Metaheurística				
	SA	VNS	AG10	MA_M10_VNS15	MA_M15_VNS15
P1	60.201	54.5	**55.135**	54.884	55.201
P2	49.634	**44.567**	45.467	45.267	45.267
P3	61.136	**54.517**	55.268	55.251	55.051
P4	50.617	**44.55**	45.684	45	45.267
P5	62.469	**55.551**	56.484	55.834	55.684
P6	50.317	**41.317**	42.934	42.801	42.317
M1	93.58	**72.869**	78.336	77.862	77.472
M2	68.186	**48.377**	52.51	51.587	52.427
M3	92.856	**75.561**	77.496	77.104	78.063
M4	63.518	52.269	53.203	**51.803**	52.885
M5	100.18	82.626	87.402	**85.279**	85.496
M6	67.494	**47.526**	52.743	52.318	50.925
G1	116.81	**94.125**	102.61	102.805	101.975
G2	68.245	**53.945**	56.965	57.54	58.075
G3	114.93	114.93	**102.565**	103.62	103.115
G4	67.33	**54.71**	56.74	57.505	58.1
G5	116.25	**95.691**	102.69	103.15	102.04
G6	63.23	**44.8**	50.375	49.835	52.165

Tabla 50. Comparación de los resultados obtenidos para reducción de tiempo en la comparación de metaheurísticas y el algoritmo memético

Tipo de Proyecto	Valor de Friedman	Prob Chi^2	K – 1	Punto Crítico	Se rechaza
Pequeño	17.96	0.001	4	9.4877	H_0
Mediano	17.76	0.001	4	9.4877	H_0
Grande	12.28	0.015	4	9.4877	H_0

Tabla 51. Resultados de la prueba de Friedman para reducción de tiempo en la comparación de metaheurísticas y el algoritmo memético

Algoritmo	Tenor	Valor	¿Hay Dif.?
MA_M10_VNS15	0.05	0.06680720126	No
MA_M15_VNS15	0.025	0.04456546275	No
AG10	0.016666	0.00255513033	Si
SA	0.0125	0.00003167124	Si

Tabla 52. Resultados de la prueba de Holm para reducción de tiempo (Proyectos pequeños) en la comparación de metaheurísticas y el algoritmo memético. Algoritmo de Control: VNS

Algoritmo	Tenor	Valor	¿Hay Dif.?
MA_M10_VNS15	0.05	0.21185539858	No
MA_M15_VNS15	0.025	0.03593031911	No
AG10	0.016666	0.0046618802	Si
SA	0.0125	0.00007234804	Si

Tabla 53. Resultados de la prueba de Holm para reducción de tiempo (Proyectos medianos) en la comparación de metaheurísticas y el algoritmo memético. Algoritmo de Control: VNS

Algoritmo	Tenor	Valor	¿Hay Dif.?
AG10	0.05	0.30853753873	No
MA_M15_VNS15	0.025	0.13566606095	No
MA_M10_VNS15	0.016666	0.04456546276	No
SA	0.0125	0.00068713794	Si

Tabla 54. Resultados de la prueba de Holm para reducción de tiempo (Proyectos grandes) en la comparación de metaheurísticas y el algoritmo memético. Algoritmo de Control: VNS

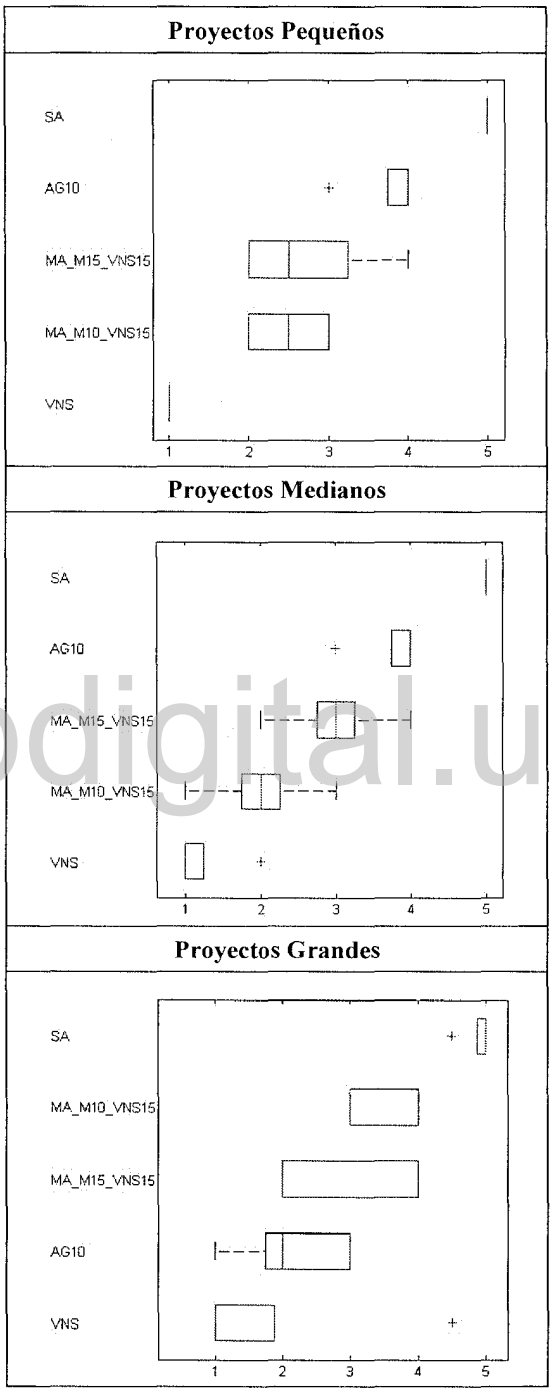


Tabla 55. Diagramas de Cajas y Bigotes de la comparación de metaheurísticas y el algoritmo memético para reducción de tiempo

Finalmente, dado que cada uno de los casos de prueba estipulados representan instancias de un mismo problema, a continuación en la Tabla 56, se muestran los diagramas de cajas y bigotes de la comparación de todas las estrategias implementadas para cada enfoque de la función objetivo: reducción de costos y reducción de tiempo (asignando o no desarrolladores expertos), los cuales fueron obtenidos a partir del resumen de las medias de cada uno de dichos casos. A partir de este resumen, se puede observar nuevamente al VNS como el algoritmo con mejores resultados para todas las instancias, y las diferentes versiones del genético con resultados equiparables al mismo.

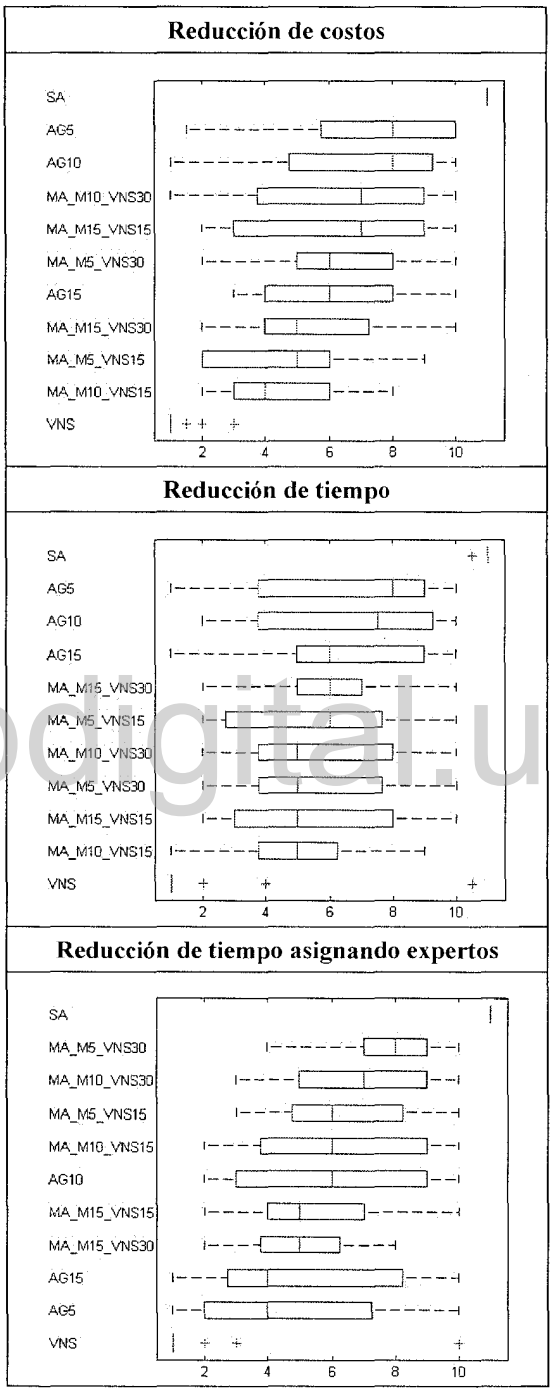


Tabla 56. Diagramas de Cajas y Bigotes de comparación de las estrategias implementadas para todos los casos de prueba

Capítulo 6. Conclusiones

En esta investigación se ha propuesto la implementación de metaheurísticas para dar solución al problema de planificación de proyectos de software. Las mencionadas metaheurísticas resultaron ser estrategias mediante las cuales efectivamente es posible ofrecer solución a dicho problema, el cual fue abordado como un problema de optimización combinatoria a partir de la representación del problema de planificación de proyectos. Los parámetros del problema se sustentaron en información suministrada por la compañía Caniatech y se estimaron mediante la aplicación metodologías ágiles de desarrollo de proyectos de software, específicamente en cuanto a la indicación de la duración de las tareas del proyecto a través de puntos de complejidad y las restricciones propias de los proyectos de planificación de software.

Las metaheurísticas implementadas fueron el recocido simulado (SA), la búsqueda con vecindario variable (VNS) y un algoritmo genético (GA) respectivamente. Éstas fueron seleccionadas como estrategias sin memoria y de tipo poblacional o de trayectoria, con el objeto de comparar los resultados obtenidos con métodos cuya exploración del espacio de búsqueda sea diferente entre sí. Por otra parte, para la ejecución de las pruebas de cada uno de los algoritmos realizados se definieron un conjunto de escenarios de prueba, los cuales se indican en primer lugar a partir del tamaño del proyecto de software y en segundo lugar a partir de las

variantes que puede tener dependiendo de la cantidad de desarrolladores con los que puede contar el equipo de desarrollo, y las habilidades o experiencia de cada uno de ellos. Dicho conjunto se conformó en total por 18 casos de prueba, los cuales fueron evaluados por cada una de las metaheurísticas desarrolladas, para cada enfoque de la función objetivo: reducción de tiempo, reducción de costos, o reducción de tiempo asignando lo desarrolladores con más experiencia para cada tarea. Es importante destacar que los casos de prueba fueron diseñados a partir de proyectos de software reales, con miras a evaluar posteriormente el progreso de las planificaciones creadas.

Para todos los casos de prueba, el VNS y GA arrojaron resultados equiparables, debido a que la consideración de los cambios factibles de desarrolladores por cada tarea permite realizar un recorrido más amplio por las posibles planificaciones de un proyecto determinado. Asimismo, la hibridación de metaheurísticas para la implementación del algoritmo memético estructurado a partir del algoritmo genético y del VNS permitió comparar los resultados obtenidos con una estrategia más robusta con la que se obtuvieron mejores resultados y menos dispersos que los obtenidos a través del algoritmo genético.

Capítulo 7. Trabajo a Futuro

El trabajo a futuro de esta investigación está definido principalmente a partir de dos líneas de acción. En primer lugar, resulta necesario implementar otras metaheurísticas de diversos tipos para dar solución a la planificación de proyectos de software. La selección realizada en esta investigación se basó en mecanismos sin memoria, es fundamental realizar la comparación de los resultados a obtener a partir de estrategias con memoria o de otras clasificaciones, conformando un abanico más amplio de opciones. Asimismo, la inclusión de estas nuevas estrategias permitirá formular diferentes mecanismos de hibridación.

Por otra parte, la implantación y seguimiento de las planificaciones generadas a través de las metaheurísticas implementadas, en empresas de desarrollo de software permitirá determinar la efectividad de dichas planeaciones. De esta manera, será posible incluir otros factores que puedan afectar la planificación y que resulten fundamentales para el éxito del proyecto. Adicionalmente, dado que el problema de planificación en general es naturalmente dinámico y multiobjetivo, se requiere definir nuevas estrategias algorítmicas para abordarlo y extrapolarlo al campo al campo de la planificación de software, podría abordarse por ejemplo, mediante técnicas de computación evolutiva interactiva o programación dinámica.

Referencias

Abdel, T., & Madnick, S. (1983). The dynamics of software project scheduling. *Communications of the ACM, ACM , 26*, 340-346.

Beltran, J., Brito, J., Campos, C., Garcia, I., Garcia, M., Cabrera, J., y otros. (2004). Metaheurísticas: una revisión actualizada. *Grupo de Computación Inteilgente*

Biolchini, J., Gomes, P., Cruz, A., & Horta, G. (2005). *Systematic Review in Software Engineering.* Reporte Técnico, Systems Engineering and Computer Science Department, Rio de Janeiro.

Blazewicz, J., & Lenstra, J. (1983). *Scheduling subject to resource constraints: classification and complexity.* Stichting Matematisch Centrum.

Brito, J., Campos, C., García, F., García, M., Belén, B., Moreno, J., y otros. (20 de Junio de 2004). Metaheurísticas: Una revisión actualizada. *Grupo de computación inteligente. Universidad de La Laguna .*

Caniatech. (2012). *Caniatech*. Obtenido de http://caniatech.com/

Cervantes, M. (2010). *Nuevos métodos meta heuristicos para la asignación eficiente, optimizada y robusta de recursos limitados.* Departamento de Sistemas Informáticos y Computación. Valencia: Universidad Politécnica de Valencia.

Chicano, J. (2007). *Metaheurísticas e Ingeniería del Software.* Málaga: Tesis Doctoral. Universidad de Málaga.

Cotta, C. (2007). Una Visión General de los Algoritmos Genéticos. *Dept. Lenguajes y Ciencias de la Computación, ETSI Informática* .

Dowsland, K., & Adenso, B. (2003). Diseño de Heurística y fundamentos del recocido simulado. *Revista Iberoamericana de Inteligencia Artificial , 7* (19), 93-102.

Duncan, W. (2004). *A guide to the project management body of knowledge.* USA: Project Management Institute.

Durán, S. (2003). Puntos por Función. Una métrica estándar para establecer el tamaño del software. *Boletín de Política Informática* (6), 41-52.

García, M. (2000). Optimización Combinatoria. *Números , 43-44*, 115-120.

García, S., Molina, D., Herrera, F., & Lozano, M. (2009). A Study on the Use of Non-Parametric Tests for Analyzing the Evolutionary Algorithms Behavior: A Case Study on the CEC´2005 Special Session on Real Parameter Optimization. *A case study on the CEC'2005 Special Session on Real Parameter Optimization Journal of Heuristics, Springer Netherlands , 15*, 617-644.

Gestal, M. (2003). Sistemas adaptativos y bioinspirados en inteligencia artificial. *Depto. Tecnologías de la Información y las Comunicaciones, Universidade da Coruña* .

Glover, F. (1986). Future Paths for integer programming and links to artificial intelligence. *Computer & Operations Research* , 533-549.

Goldberg, D. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison-Wesley.

Grompone, J. (1996). *Gestion de proyectos de software* (Primera ed.). Montevideo, Uruguay, Uruguay: La Flor del Itapebi.

Hansen, P., & Mladenovic, N. (2003). Variable Neighbourhood Search. *Inteligencia Artificial* (19), 77-92.

Junk, W. (17 de Abril de 2000). The Dynamic Balance Between Cost, Schedule, Features and Quality in Software Development Projects. *Computer Science Dept., University of Idaho* , SEPM-001.

Kniberg, H. (2007). *SCRUM y XP desde las trincheras.* USA: C4Media Inc.

Krasnogor, N., & Smith, J. (2005). A Tutorial for Competent Memetic Algorithms: Model, Taxonomy and Design Issues. *IEEE Transactions on evolutionary computation , A* (B).

Larrañaga, P., Abdelmalik, M., & Iñaki, I. (2003). Algoritmos de Estimación de Distribuciones en Problemas de Optimización Combinatoria. *Inteligencia Artificial, Revista Iberoamericana , 7*, 149-168.

Larrañaga, P., Kuijpers, C. M., Murga, R. H., & Inza, I. (1999). S. Genetic Algorithms for the Travelling Salesman Problem: A Review of Representations and Operators. *Articial Intelligence Review , 13*, 129-170.

Lili L., D. W. (2011). A new differential evolution algorithm for dynamic scheduling problems with variant job weights. *Control and Decision Conference (CCDC)* , 274-278.

Luna, F., Chicano, F., & Alba, E. (2012). Robust solutions for the software project scheduling problem: a preliminary analysis. *Int. J. Metaheuristics* , 56-79.

Martí, R. (2003). Procedimientos Metaheurísticos en Optimización Combinatoria. *Matemátiques. Universitat de Valencia , 1*, 3-62.

Martínez, C. (2011). *Metaheurísticas hibridas aplicadas al Problema de Ruteo de Arcos Capacitados.* Buenos Aires: Universidad de Buenos Aires.

Melián, B., Moreno, J., & Moreno, M. (2003). Metaheuristics: A global view. *Inteligencia Artificial, Revista Iberoamericana de Inteligencia Artificial* (19), 7-28.

Mian, P., Conte, T., Natali, A., Biolchini, J., & Travassos, G. (2005). Experimental Software Engineering Latin American Network. *UFRJ - Computer Science Department - Brazil* .

Moscato, P. (1989). On Evolution, Search, Optimization, Genetic Algorithms. *Caltech Concurrent Computation Program. California Institue of Technology* .

Pinedo, M. (1983). Stochastic scheduling with release dates and due dates. *Oper. Res.* , *31*, 559-572.

Rivero, D., Montilva, J., Granados, G., Barrios, J., Besembel, I., & Sandia, B. (s.f.). La Industria de Software en Venezuela: Una Caracterización de su Recurso Humano. *Grupo de Investigación en Ingeniería de Datos y Conocimiento. Universidad de los Andes, Facultad de Ingeniería* .

Sarasola, B., Doerner, K., & Alba, E. (2012). Un algoritmo de búsqueda en vecindario variable para la asignación de rutas a vehículos con pedidos dinámicos. (U. d. Málaga, Ed.) *Departamento de Lenguajes y Ciencias de la Computación, ETSI Informática, Universidad de Málaga* .

Talbi, E.-G. (2009). *Metaheuristics: From Design to Implementation.* Canada: Wiley.

Vengerov, D., Mastroleon, L., Murphy, D., & Bambos, N. (2007). Adaptive data-aware utility-based scheduling in resource-constrained systems. *Sun Microsystems, Inc.* , SMLI TR-2007-164.

Wolpert, D. H., & Macready, W. G. (1997). No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation* , 67-82.

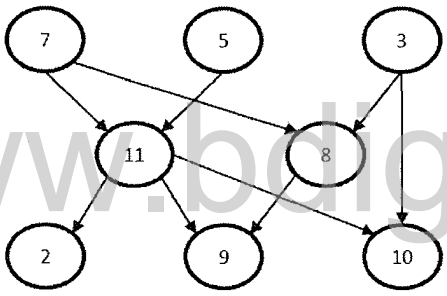
Yavuz, M., Akcali, E., & Tufekci, S. (2006). A Hybrid Meta-Heuristic for the Batching Problem in Just-In-Time Flow Shops. *Journal of Mathematical Modelling and Algorithms, Springer Netherlands* , 371-393.

ANEXOS

ANEXO A. Recorrido Topológico de un Grafo

Tomado de "Introducción a la Teoría de Grafos".
Matemáticas Discretas. Chacón, José.

El recorrido u ordenación topológica de un grafo G es una ordenación lineal de todos los nodos de G que conserva la unión entre vértices del grafo G original. Este tipo de recorridos solo es posible para grafos acíclicos. Es ampliamente utilizado para la representación de problemas con relaciones de precedencia, como en el caso de la planificación de proyectos.



Grafo "G"

Recorridos Topológicos del Grafo "G"

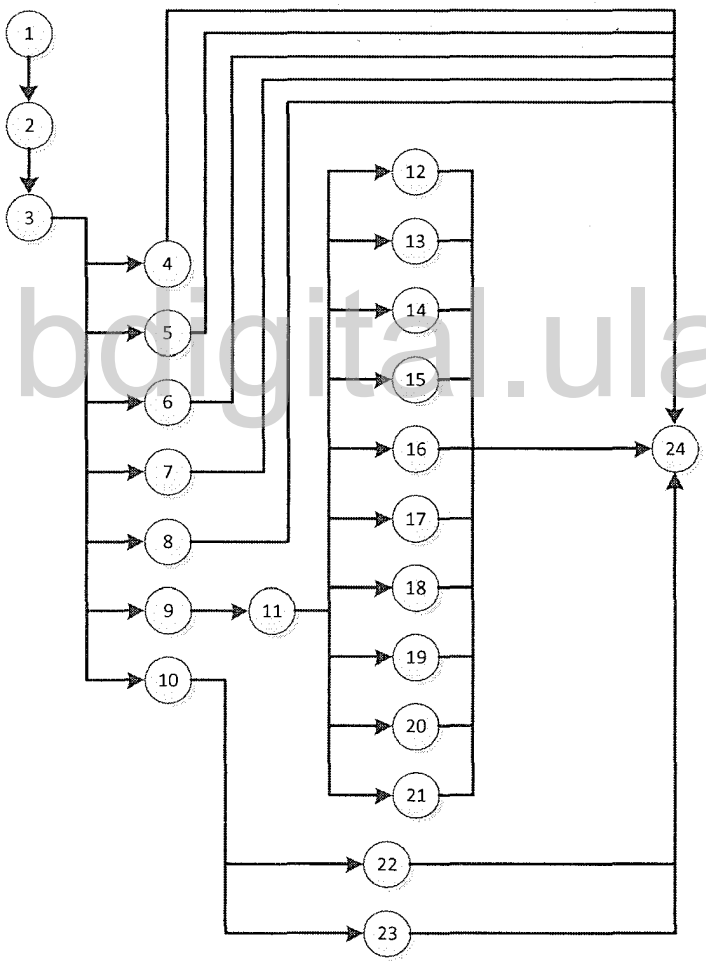
7, 5, 3, 11, 8, 2, 9, 10
3, 5, 7, 8, 11, 8, 9, 10
3, 7, 8, 5, 11, 10, 2, 9
5, 7, 3, 8, 11, 10, 9, 2
7, 5, 11, 3, 10, 8, 9, 2
7, 5, 11, 2, 3, 8, 9, 10

En el ejemplo anterior, existen seis diferentes recorridos topológicos cada uno de los cuales respeta las relaciones de precedencia indicadas a través de los arcos del grafo. De esta manera, si el grafo "G" representara las precedencias a partir del recorrido topológico es posible determinar el orden en el que pueden ser realizadas culminando en primer lugar aquellas que anteceden a las tareas subsiguientes. Adicionalmente este recorrido permite conocer el nivel de profundidad de cada nodo, indicado por los nodos que se deben recorrer desde el punto de inicio para llevar a él. Así, si el nodo no tiene precedentes corresponde al nivel cero, y si es el nodo final del grafo corresponde al recorrido más largo que se debe recorrer para llegar a éste.

ANEXO B. Grafos de precedencia de tareas de los casos de prueba

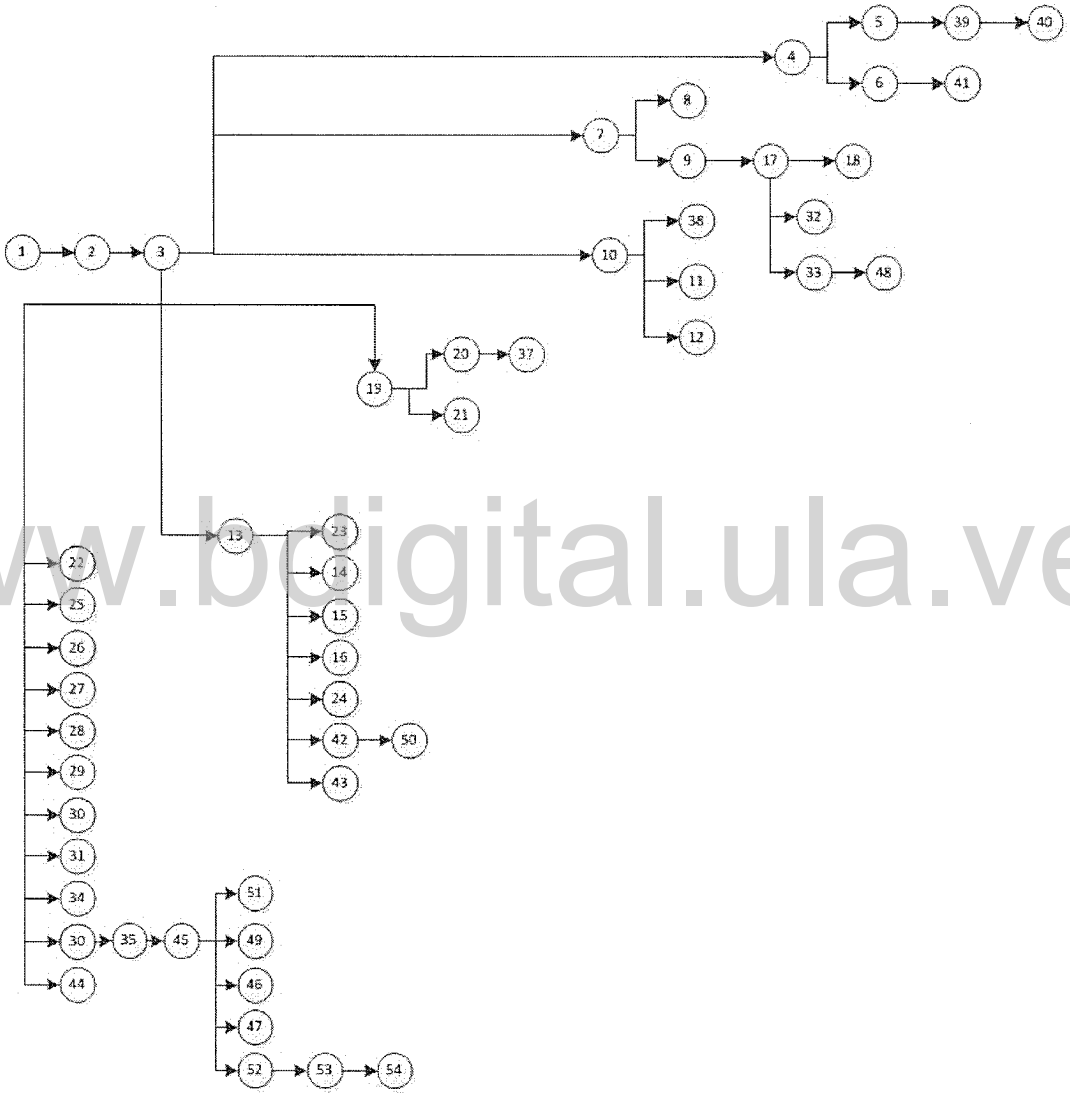
Anexo B. 1

Grafo de Precedencia de Tareas TPG – Proyectos Pequeños



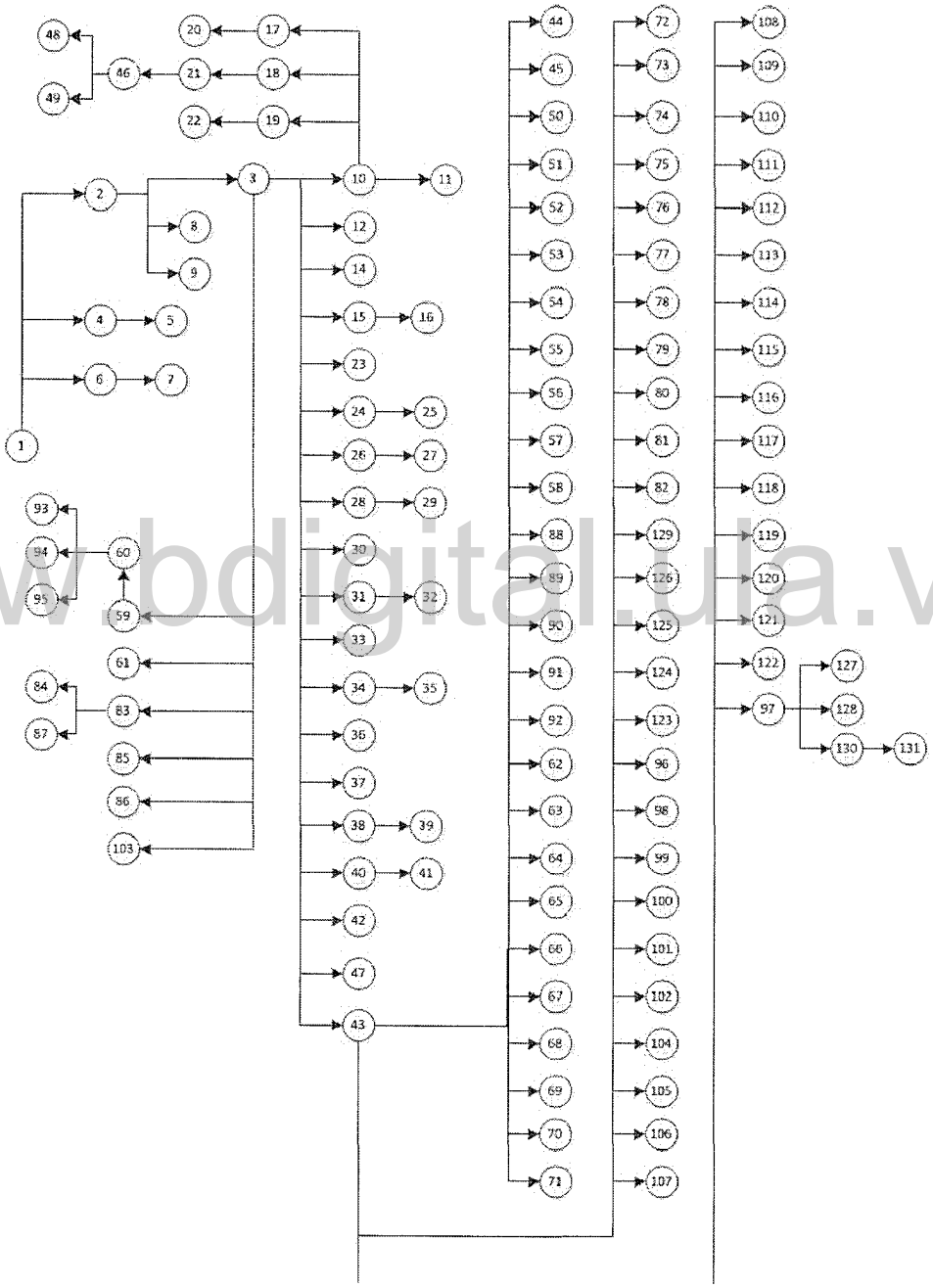
Anexo B. 2

Grafo de Precedencia de Tareas TPG – Proyectos Medianos



Anexo B. 3

Grafo de Precedencia de Tareas TPG – Proyectos Grandes



ANEXO C. Desviaciones estándar de las pruebas realizadas

Anexo C. 1

Desviación de estándar para la reducción de costo con metaheurísticas simples

Caso de Prueba	Metaheurística				
	SA	VNS	AG5	AG10	AG15
P1	208.6864	29.6742	195.4143	119.9985	207.8982
P2	277.9284	153.5561	151.9984	192.5401	142.8554
P3	194.8174	56.7520	139.1904	159.9676	173.0730
P4	289.0484	57.9221	223.2501	204.7079	258.6114
P5	173.9553	10.5409	70.9558	115.4701	102.3678
P6	132.6382	52.0388	129.3906	131.2824	97.0721
M1	216.6882	151.0778	203.9292	169.1657	181.9153
M2	265.2837	235.2187	164.2677	110.2317	332.9622
M3	288.3468	99.2655	196.4653	267.2609	133.4603
M4	386.7959	319.3844	251.0932	229.4950	262.7010
M5	197.4635	54.1685	191.2718	139.8495	207.7472
M6	256.7300	181.4375	232.9262	214.4699	145.9892
G1	168.9337	83.0136	83.0136	124.5072	136.6385
G2	293.8134	135.2382	228.6001	179.2135	229.0489
G3	203.1264	125.4171	91.1196	120.6927	150.6995
G4	334.7044	247.0215	158.4076	193.1679	208.2528
G5	140.2914	130.0224	113.2932	158.3640	139.9109
G6	135.3420	128.8037	208.1190	182.2140	182.2466

Anexo C. 2

Desviación de estándar para la reducción de tiempo con metaheurísticas simples

Caso de Prueba	Metaheurística				
	SA	VNS	AG5	AG10	AG15
P1	3.4760	0.01	0.7345	0.6431	0.6849
P2	1.4862	0.1612	0.9920	0.7658	0.8882
P3	1.9549	0.0538	0.5827	0.5515	0.4357
P4	2.4882	0.1581	0.7509	0.8366	0.3703
P5	2.6517	0.0821	1.1094	0.8031	0.3374
P6	3.6491	0.7344	0.7345	0.8834	0.7328
M1	4.2459	1.7217	3.8350	3.2243	2.7676
M2	2.4365	2.0849	2.3623	2.6036	2.0522
M3	2.4365	2.0849	2.3623	2.6036	2.0522
M4	3.2361	1.6007	3.8602	4.0207	3.2058
M5	4.2957	1.4568	2.5199	2.8437	1.6474
M6	5.6184	3.3189	3.6787	2.7140	1.9964
G1	3.0575	2.1822	2.5287	2.0990	2.5184
G2	4.0909	1.1427	3.0116	2.2668	1.8575
G3	1.3514	1.3514	3.0560	2.7172	3.2302
G4	3.6646	1.4942	1.7820	1.8199	2.2017
G5	4.2358	2.3945	3.2736	2.1001	3.2803
G6	3.7904	1.3530	4.3166	2.5007	1.7261

Anexo C. 3

Desviación de estándar para la reducción de tiempo asignando desarrolladores expertos con metaheurísticas simples

Caso de Prueba	Metaheurística				
	SA	VNS	AG5	AG10	AG15
P1	4.8569	0.0126	0.8950	0.7593	0.6984
P2	1.8569	0.2123	1.0251	0.9685	0.9781
P3	2.3567	0.0689	0.8475	0.6314	0.5123
P4	3.0123	0.2156	0.8569	0.7412	0.5216
P5	2.8561	0.0985	1.0563	0.0756	0.4586
P6	3.8549	0.6521	0.7676	0.9588	0.8012
M1	5.0141	1.6232	3.0456	3.5656	3.0101
M2	2.9687	1.0789	2.5678	2.7456	2.2134
M3	5.654	1.987	2.5678	2.5014	2.0589
M4	3.5621	1.4568	3.9856	4.1542	4.2356
M5	5.0147	1.2306	2.7895	2.6321	1.9854
M6	5.8956	3.5684	3.8989	3.6597	4.5067
G1	4.0621	2.0156	2.3689	2.5656	2.8964
G2	4.9871	1.0506	3.5151	2.5412	1.9752
G3	1.8956	1.2149	3.6812	2.1525	2.8956
G4	4.0521	1.3287	1.9896	1.5674	2.0556
G5	4.6154	2.0163	3.6394	2.0105	3.1056
G6	3.9631	1.1032	3.9863	2.3014	2.2841

Anexo C. 4

Desviación de estándar para la reducción de costo con el algoritmo memético

Caso de Prueba	Algoritmo Memético					
	MA_M5_VNS15	MA_M5_VNS30	MA_M10_VNS15	MA_M10_VNS30	MA_M15_VNS15	MA_M15_VNS30
P1	109.3900	77.6466	89.2441	86.3492	84.8157	101.2071
P2	151.8270	163.6896	145.2877	120.9517	172.6014	128.3305
P3	107.3925	115.1427	134.5593	106.7977	366.0375	75.6974
P4	172.4069	148.3434	159.0049	142.1025	367.0314	86.1143
P5	28.3823	45.3382	44.4878	28.8675	93.6898	33.7474
P6	154.3544	66.7254	115.8889	96.7933	339.7077	140.4059
M1	163.5450	99.4416	200.2504	139.6564	138.8349	210.1316
M2	223.7573	197.7729	287.7926	241.6706	197.5676	303.5850
M3	110.7948	114.8596	160.3122	136.4917	152.6534	169.2931
M4	340.2554	301.8797	230.7068	227.1602	236.8489	351.5476
M5	91.7528	136.4741	73.6939	341.7794	157.3874	58.7281
M6	142.3420	228.3268	225.3922	139.0094	178.8516	166.0715
G1	70.0937	136.2290	107.2267	124.6108	67.8484	119.2770
G2	227.5568	189.8986	212.8279	224.8018	249.9504	113.9639
G3	139.9228	137.5240	113.0315	151.0245	136.5927	130.3936
G4	266.1408	157.6646	213.3888	178.5095	240.7952	169.7907
G5	74.2649	86.1447	149.3905	92.1593	137.0323	72.7897
G6	112.3753	143.3339	136.7607	122.4106	177.5740	131.5584

Anexo C. 5

Desviación de estándar para la reducción de tiempo con el algoritmo memético

Caso de Prueba	Algoritmo Memético					
	MA_M5_VNS15	MA_M5_VNS30	MA_M10_VNS15	MA_M10_VNS30	MA_M15_VNS15	MA_M15_VNS30
P1	0.3693	0.3059	0.3611	0.3436	0.5540	0.4974
P2	0.3156	0.4214	0.6295	0.4249	0.4791	0.4084
P3	0.2583	0.2642	0.3538	0.5686	0.4583	0.3376
P4	0.6325	0.4830	0.4714	0.4378	0.5170	0.4471
P5	0.4025	0.3498	0.3515	0.2486	0.5249	0.2778
P6	0.7528	0.5675	0.7521	0.7756	0.4742	0.7916
M1	2.9288	1.9613	2.5924	2.4272	2.3746	2.0647
M2	1.7754	2.6531	2.1200	1.6117	2.6485	1.8201
M3	2.2430	2.4853	2.4735	2.7998	2.9434	2.0498
M4	1.8039	1.9823	1.9522	1.6148	1.6679	1.5773
M5	1.6592	2.2408	1.8134	2.2180	2.2053	2.0951
M6	1.6796	1.3704	2.4876	1.6833	1.6959	2.3877
G1	2.0897	1.5451	2.2926	2.9020	3.5083	1.7470
G2	1.4763	1.3898	1.5567	1.4654	1.2601	1.4468
G3	1.8489	2.4966	2.1370	2.4922	1.7148	1.9746
G4	2.2843	1.6670	2.1455	1.3113	1.4823	1.1200
G5	2.0845	2.7194	2.0079	2.4916	2.9164	1.5050
G6	2.6759	2.4848	2.2174	1.6765	2.3153	2.6106

Anexo C. 6

Desviación de estándar para la reducción de costo asignando desarrolladores expertos con el algoritmo memético

Caso de Prueba	Algoritmo Memético					
	MA_M5_VNS15	MA_M5_VNS30	MA_M10_VNS15	MA_M10_VNS30	MA_M15_VNS15	MA_M15_VNS30
P1	1.3957	0.7593	0.6964	0.5838	0.6964	0.6483
P2	0.9271	1.3239	0.9876	0.7783	1.6799	1.2792
P3	1.0499	0.6827	0.8139	0.9179	0.4537	0.7796
P4	1.5048	1.3211	2.0847	1.4994	2.1540	0.8095
P5	3.4663	3.0618	2.9955	3.0665	4.7011	3.6471
P6	1.8421	2.8871	4.0871	2.7655	3.1307	2.7910
M1	2.0796	1.4704	3.3730	1.5921	2.5208	2.2584
M2	2.4097	1.7938	5.4134	3.0594	3.9189	3.2195
M3	3.0001	2.0470	2.5218	3.5453	2.4542	2.1516
M4	3.8240	2.2062	4.3693	2.1691	2.3555	1.7851
M5	2.2378	2.5419	2.0975	1.7079	1.0337	2.1325
M6	1.7272	1.3704	2.4876	1.6833	1.6959	2.3877
G1	2.0897	1.5451	2.2926	2.9020	3.5083	1.7470
G2	1.4763	1.3898	1.5567	1.4654	1.2601	1.4468
G3	1.7543	2.4966	2.1370	2.4922	1.7148	1.9746
G4	2.2843	1.6670	2.1455	2.4241	1.3113	1.1200
G5	2.0845	2.7194	2.0079	2.4916	2.9164	1.5050
G6	2.6759	2.4848	2.2174	1.6765	2.3153	2.6106

Anexo C. 7

Desviación de estándar para la reducción de tiempo con diferentes versiones del AG (Análisis de Sensibilidad)

Caso de Prueba	Algoritmo Genético								
	GA_A	GA_B	GA_C	GA_D	GA_E	GA_F	GA_G	GA_H	GA_I
P1	1.07	2.18	1.08	1.08	0.57	1.82	1.16	2.04	1.06
P2	5.39	4.65	4.85	4.87	5.17	4.75	4.97	4.55	4.66
P3	1.43	2.14	1.05	2.02	1.35	1.08	1.58	1.98	0.67
P4	2.66	2.33	2.52	2.23	3.33	3.06	1.73	3.45	2.15
P5	0.47	1.34	1.34	0.95	1.18	0.60	0.50	0.56	0.42
P6	1.49	3.08	2.13	2.04	0.86	1.50	3.09	2.63	1.70
M1	2.86	2.27	3.27	4.94	2.59	3.53	3.63	4.96	2.98
M2	7.58	9.09	8.96	8.10	7.84	9.17	8.65	5.90	8.98
M3	2.58	3.53	3.87	2.45	2.86	3.05	3.28	2.21	2.88
M4	4.60	6.50	5.41	5.96	6.76	4.99	6.07	6.41	9.14
M5	1.19	2.23	2.39	2.37	2.51	2.45	3.10	3.24	2.77
M6	6.34	6.26	6.46	4.50	5.28	6.62	6.93	5.93	8.77
G1	1.88	2.53	3.03	1.64	2.68	1.29	2.68	1.99	2.68
G2	12.67	13.97	14.18	13.48	13.50	13.12	12.75	12.96	13.01
G3	2.47	2.12	2.68	2.50	2.40	1.72	2.60	3.75	2.08
G4	3.75	4.42	4.43	2.47	3.83	4.03	4.53	4.32	3.75
G5	2.85	2.06	3.31	1.85	2.93	2.32	2.49	2.98	2.78
G6	4.38	3.95	3.98	4.25	5.08	3.75	4.96	4.78	2.88

Anexo C. 8

Desviación de estándar para la reducción de tiempo con diferentes versiones del AM (Análisis de Sensibilidad)

Caso de Prueba	Algoritmo Memético					
	MA_A	MA_B	MA_C	MA_D	MA_E	MA_F
P1	1.73	0.79	2.23	1.83	1.46	1.45
P2	4.54	5.11	4.69	5.16	5.60	4.48
P3	1.50	1.31	1.46	1.50	2.27	1.57
P4	2.33	2.54	2.16	2.87	3.92	2.51
P5	1.05	1.02	0.69	1.17	0.96	0.36
P6	3.17	2.11	2.12	2.77	2.19	1.45
M1	4.32	2.58	3.36	3.36	2.94	2.92
M2	8.68	8.39	8.79	8.20	6.82	6.76
M3	2.67	2.36	2.89	2.69	3.88	3.39
M4	5.55	5.10	6.17	6.07	7.85	4.93
M5	2.08	2.09	2.36	2.97	1.74	1.63
M6	4.77	5.33	4.89	5.46	4.02	5.51
G1	2.30	2.38	2.06	1.65	1.76	2.01
G2	14.27	13.33	13.89	13.80	14.67	13.65
G3	2.06	2.04	2.54	2.49	1.92	1.16
G4	3.70	2.73	1.65	3.60	3.82	4.14
G5	1.44	2.40	1.60	1.36	2.01	1.77
G6	4.25	4.09	2.89	3.56	4.41	2.80