

UNIVERSIDAD NACIONAL MAYOR DE SAN MARCOS Fundada en 1551

FACULTAD DE INGENIERÍA DE SISTEMAS E INFORMÁTICA

UNIDAD DE POST GRADO

Un algoritmo GRASP para resolver el problema de la programación de tareas dependientes en máquinas diferentes (task scheduling)

para optar el Grado Académico de: **MAGÍSTER EN INGENIERÍA DE SISTEMAS** con la mención en
Ingeniería de Software

AUTOR

MANUEL FRANCISCO TUPIA ANTICONA

ASESOR Dr. DAVID MAURICIO SÁNCHEZ

LIMA - PERÚ 2005

..	1
AGRADECIMIENTOS .	3
RESUMEN .	5
ABSTRACT .	7
Capítulo I: Introducción . .	9
1.1 Panorama general del proyecto . .	9
1.2. Justificación de la aplicación de un algoritmo meta heurístico en la planificación de tareas de un proyecto de desarrollo de software. .	10
1.3. La Administración de Proyectos y la planificación de recursos . .	11
1.4. Aportes propuestos . .	12
1.5. Metodología sobre la que se aplican los algoritmos propuestos . .	12
1.6. La planificación como antecedente del task scheduling. .	13
1.7. La Meta Heurística GRASP .	13
1.8. Organización y estructura de la tesis .	14
Capítulo II Marco teórico .	15
2.1. La planificación como problema de optimización y antecedente del task scheduling. .	15
2.2. Estado del arte del problema de la Planificación de Tareas .	16
2.2.1. Objetivos del scheduling .	17
2.2.2. Variantes del problema del scheduling . .	17
2.3. Análisis de la variante del task scheduling propuesta . .	19
2.4. Métodos existentes para resolver el problema del scheduling. .	19
2.4.1. Métodos heurísticos y meta heurísticos planteados por variante .	20
2.5. Metodología sobre la cual se aplicarán los algoritmos: RUP. . .	22
2.5.1. Característica 1: RUP está dirigido por medio de casos de uso (use case) .	23
2.5.2. Característica 2: RUP está centrado en la arquitectura . .	23
2.5.3. Característica 3: RUP es iterativo e incremental .	24

2.5.4. Otra característica: RUP también es un producto .	24
2.5.5. Los recursos convertidos en <i>máquinas</i> .	25
2.5.6. Las fases de RUP . .	26
2.6. Aplicación práctica de RUP en la planificación de proyectos . .	26
2.6.1. Programando los objetivos definidos del proyecto RUP .	27
2.6.2. Empleo de las fases RUP y su relación con la programación de tareas .	28
2.6.3. Calendarizar el proyecto en cuanto al esfuerzo estimado a realizarse por fase .	29
2.6.4. Priorizar riesgos y manejo de restricciones . .	29
2.6.5. Definición de hitos y de iteraciones dentro de cada fase .	30
2.6.6. Las dos fases iniciales RUP: el problema de planificar las operaciones involucradas . .	30
2.7. Flujos de trabajo o disciplinas: planificando recursos involucrados . .	32
2.7.1. Modelado de negocios (business modeling) . .	34
2.7.2. Captura de requisitos .	35
2.8. Algunas metodologías de planificación de proyectos complementarias a RUP: estado del arte de la planificación de proyectos de Software .	37
2.8.1. Planificación en proyectos usando COST: comercial off-the-shelf .	37
2.8.2. UED: Unifying Enterprise Development Teams .	38
2.8.3. IBM-Rational Unified Process: IBM – RUP .	38
2.8.4. eXtreme Programming: RUP para pequeños proyectos .	38
2.8.5. MSP: Integrando Managing Successful Programmes con RUP .	39
2.8.6. Estimación de proyectos con IBM Rational SUMMIT Ascendant .	41
2.8.7. Project Portfolio Management (PPM): alineando negocios con Tecnologías de Información -TI .	42
2.9 Conceptos generales sobre heurísticas y meta heurísticas . .	43
2.9.1. Fase de construcción GRASP .	45
2.9.2. Fase de mejoría de la solución GRASP . .	47
Capítulo III: Un modelo matemático de optimización para el problema del task scheduling en la planificación de los recursos en los proyectos de Ingeniería de Software . .	49
3.1. Elementos del modelo. Consideraciones generales .	49

3.2. Variables de decisión del modelo .	50
3.3. Restricciones del modelo .	50
3.4. El modelo de optimización matemática para la variante desarrollada . .	51
3.4.1. Comentarios sobre el modelo presentado: relajación . .	52
3.5. Importancia del modelo matemático .	52
Capítulo IV: Algoritmos Propuestos . .	55
4.1. Alcances y limitaciones de la solución a plantearse .	55
4.2. Estructuras de datos empleadas .	56
4.3. Lineamientos generales del algoritmo voraz propuesto .	57
4.3.1. Ejemplo de aplicación: criterio voraz para la selección de la mejor tarea .	58
4.3.2. Ejemplo de aplicación del criterio voraz para la selección mejor recurso ejecutor . .	58
4.3.3. Presentación del Algoritmo Voraz propuesto . .	59
4.3.4. Comentarios sobre el algoritmo presentado .	60
4.3.5. Ejemplo de aplicación del algoritmo voraz . .	60
4.4. Algoritmo GRASP propuesto: lineamientos generales .	63
4.4.1. Criterio GRASP para la selección aleatoria de la mejor tarea a ejecutar. .	64
4.4.2. Criterio GRASP para la selección aleatoria del mejor recurso ejecutor. .	64
4.4.3. Presentación del Algoritmo GRASP – fase de Construcción .	64
4.4.4. Comentarios sobre el algoritmo GRASP – fase Construcción presentado .	65
4.4.5. Ejemplo de aplicación del algoritmo GRASP .	66
4.5. Alcance de la solución propuesta en la perspectiva de Ingeniería de Software . .	67
Capítulo V. Implementación de los algoritmos .	69
5.1. Modelado de negocios del prototipo .	69
5.2. Diagrama de actividades . .	70
5.3. Diagramas con las especificaciones de los casos de uso .	71
5.3.1. Especificación: Caso de uso Ingreso de los datos de entrada . .	71

5.3.2. Especificación: Caso de uso Ejecutar programación .	72
5.4. Diagramas de secuencia . .	72
5.4.1. Diagrama de secuencia Ingresar datos de entrada .	73
5.4.2. Diagrama de secuencia Ejecutar programación . .	73
5.5. Diagrama de clases .	74
5.6. Consideraciones sobre el ambiente de desarrollo . .	74
5.6.1. Entorno de desarrollo utilizado .	74
5.6.2. Dispositivos de almacenamiento de datos: archivos usados. .	75
5.6.3. Alcances y limitaciones del prototipo implementado .	75
5.6.4. Módulos del sistema y clases más importantes . .	75
5.7. Módulos del sistema .	77
5.7.1. Modulo de ingreso de datos de la instancia de trabajo . .	77
5.7.2. Módulo de ejecución de los algoritmos . .	80
5.7.3. Módulo de reportes y presentación de resultados . .	81
5.8. Requerimientos mínimos de hardware y software . .	83
5.8.1 Configuración de hardware mínimo .	83
5.8.2. Configuración de software mínimo. . .	83
Capítulo VI. Experimentos numéricos .	85
6.1. Preliminares .	85
6.2. Hardware y software empleados .	86
6.3. Los problemas test: proporción tareas / recursos . .	86
6.4. Calibración de las constantes de relajación α y θ . . .	87
6.4.1. Proceso de calibración para una instancia determinada .	87
6.5. Resultados de los Experimentos .	89
6.6. Calidad real de las soluciones .	90
6.6.1. Instancias usadas. Rango de tamaños .	91
6.6.2. Ejemplo de una instancia modelada en LINDO . .	91
6.6.3. Cuadro resumen de las experiencias realizadas con LINDO. . .	94
Capítulo VII. Análisis de los experimentos realizados .	95

7.1. Análisis de la eficiencia de los algoritmos entre sí .	95
7.1.1. Criterios de relajamiento empleados . .	96
7.1.2. Rendimiento de los algoritmos .	96
7.2. Análisis de eficiencia real .	97
7.2.1. Calidad de la solución versus el número de iteraciones de la fase de construcción GRASP .	97
7.2.2. Calidad de la solución versus los valores de la constante de relajación α .	98
7.3. Análisis de los resultados experimentales desde la perspectiva de la Ingeniería de Software .	99
Capítulo VIII Conclusiones .	101
8.1. Conclusiones sobre el proyecto . .	101
8.2. Conclusiones sobre los algoritmos propuestos . .	102
8.3. Aplicabilidad en la Industria Peruana .	102
8.4. Futuras investigaciones: task scheduling y optimización en general .	103
8.5. Recomendaciones Finales .	103
BIBLIOGRAFÍA .	105
ANEXO . .	111
ANEXO A: Detalle de los experimentos numéricos .	111
ANEXO B: Eficiencia de la solución GRASP .	113
ANEXO C: Modelos matemáticos usados para pruebas exactas . .	114
TEXTO COMPLETO EPUB . .	119

DEDICATORIA A los amigos que perdí y a los que se fueron

AGRADECIMIENTOS

Hay una no tan larga lista de personas a las que quisiera agradecer en este momento dado que sin su apoyo no hubiese podido realizar este proyecto.

A mi familia, porque soportó conmigo el casi interminable proceso antes de la sustentación. Ellos me dieron su incondicional apoyo y consuelo y tuvieron que soportar también, mi permanente mal humor y frustración frente a las dificultades que fueron apareciendo. La realización de este proyecto es un éxito familiar más que personal: espero que haber respondido a sus expectativas.

Al Doctor Luís Rivera y al Mag. Edgardo Palza quienes me facilitaron casi de forma exclusiva, todo el material bibliográfico que he utilizado en la elaboración de la tesis tesis. Prácticamente sin su apoyo, hubiese sido imposible realizar la investigación teórica. No tengo palabras para agradecer su desprendimiento, su diligencia y la amistad sincera que me brindaron durante la realización de este proyecto.

A la Doctora Nora La Serna, Directora de la Unidad de Post Grado de Facultad de Ingeniería de Sistemas e Informática, quien con su empeño sacó la parte administrativa de este proyecto.

A mi querida Mary Rose, compañera, alumna y futura colega por su apoyo y cariño incondicional. Supo sufrir conmigo las peripecias que me deparó este fin de proyecto. A ella mi más profundo agradecimiento y amor. Espero poder acompañarla y ayudarla el resto de su carrera y el resto de la vida también. TAM.

A Dios, por darme paciencia y por secar mis lágrimas en las noches.

A mi padre que ya está en la gloria de Krshna pero que aún me sigue cuidando. Te lo dije hace ocho años y te lo repito ahora: nos veremos pronto.

RESUMEN

La planificación de las operaciones involucradas en un proyecto de desarrollo de software ha sido un problema a superar, desde el auge del uso de metodologías que guían dicho proceso. Tanto la eficiencia como la sofisticación de los algoritmos que buscan resolver los estos ordenamientos, han ido evolucionando durante la segunda mitad del siglo XX. Al mencionado problema de ordenar tareas con dependencias se le conoce en la algorítmica como **programación de tareas o task scheduling** y es definido de la siguiente forma: dado un conjunto de tareas a ser programadas en determinado grupo de máquinas (o recursos hombre-máquina como podrían ser programadores, analistas, etc.), encontrar un orden adecuado de ejecución. Es un problema de complejidad NP-difícil por lo que se justifica el uso de métodos heurísticos para obtener soluciones aproximadas. El presente trabajo de tesis **presenta una meta heurística GRASP** para resolver la variante en donde las tareas son dependientes y los organismos ejecutores son diferentes entre sí: con esto se podrían planificar las tareas de las etapas iniciales de un proceso particular de desarrollo de software. En la tesis, se incide en la metodología RUP, y en particular en sus disciplinas de modelamiento de negocios (business modeling) y captación de requerimientos (requirement). Se han desarrollado tanto un algoritmo voraz como una meta heurística GRASP con dos parámetros de relajación, planteamiento novedoso pues hasta el momento no se había intentado resolver el problema de esta forma. Igualmente se muestra un modelo matemático para la variante específica del problema a tratar.

Para demostrar la corrección de los algoritmos, se desarrolló un prototipo que los implementa obteniéndose como resultado que el algoritmo GRASP mejora casi en un 6% los resultados del algoritmo voraz, para instancias de hasta 12500 variables involucradas.

Palabras clave: Programación de tareas, algoritmos GRASP, Desarrollo de Software, planificación de recursos en proyectos de desarrollo software, RUP.

ABSTRACT

Operation's planning for Software Development has been a complicated by-solve problem experienced since the golden age of the use methodologies whose rule those process. In which it is used, as well as in the efficiency and sophistication of the algorithms that try to solve the problems that appear in a software project, since its origin in the middle of the 20th century. The previously mentioned problem is known within algorithmic as **task scheduling** and it is defined as follows: given a group of tasks (operations) to be scheduled within a group of machines (or human resources, or human-machine resources), find an appropriate execution order. It is a NP-difficult complexity problem, so it justifies the usage of heuristic methods to obtain approximate solutions. This thesis presents a **GRASP heuristic** goal to solve the variant in which tasks are dependent and executing entities are different one from the other: now it could be possible the planning of the operation s from the inception's RUP phase. We are remarking in particular, two disciplines of RUP methodology: business modeling and requirement. Both a greedy algorithm and a GRASP heuristic goal with two relaxation parameters have been developed. Innovative proposition because until now nobody has tried to solve the problem this way. Likewise a mathematical model for the specific variant of the problem to be considered is shown.

To show efficiency of the GRASP algorithm, we developed a prototype program that executes and compares the results obtained by greedy and GRASP algorithms. The GRASP algorithm improves by 6% the results of the greedy algorithm, for instances with up to 12500 variables involved. Finally we measured the quality of these results with those of the mathematical model which would obtain the exact solution for smaller instances, taking advantage of software that solves linear programming problems: the GRASP algorithm got close to the exact result within a range of 95 to 99%, and even equaled it in some tests.

Keywords: Task scheduling, GRASP algorithm, Software development, resource planning in software projects, RUP.

Capítulo I: Introducción

1.1 Panorama general del proyecto

Se ha identificado que las metodologías de desarrollo de proyectos de software no plantean de forma adecuada, la forma cómo efectuar la planificación de las tareas y la asignación de los recursos de las fases que incluyen. Esta carencia aumenta el esfuerzo que los administradores deben realizar en el proceso de planificación; situación que se complica en particular, en las etapas iniciales donde se recaba la información que será usada en estados subsiguientes, lo que muestra una su marcada dependencia.

Partiendo de la identificación y adaptación de un problema encontrado en la planificación de operaciones dentro del entorno del desarrollo de software, se quiere cubrir la necesidad de un ordenamiento para las tareas que son ejecutadas en cada fase del ciclo de vida del producto, contemplando dependencia y heterogeneidad en el flujo de trabajo.

El presente trabajo de tesis desarrolla un algoritmo meta heurístico del tipo GRASP [FR95] (por sus siglas en inglés Greedy Randomize Adaptive Search Procedure) para resolver el problema de la **planificación de los recursos involucrados en un proceso de desarrollo de software** en donde las tareas a ejecutarse son dependientes y los recursos son de carácter heterogéneo en cuanto a su rendimiento.

1.2. Justificación de la aplicación de un algoritmo meta heurístico en la planificación de tareas de un proyecto de desarrollo de software.

La investigación ha partido de la necesidad de la planificación y ordenamiento de los recursos y las tareas en las fases iniciales de un proyecto de desarrollo de software. Dado que el problema proviene de métodos para organizar operaciones dentro de líneas de producción industrial, la mayoría de las aplicaciones existentes han tenido esa inclinación. Sin embargo, se ha podido adaptar perfectamente el concepto hacia la planificación y ordenamiento de recursos dentro de un proyecto de Ingeniería de Software porque guarda características muy similares al comportamiento de una línea de producción convencional como lo presenta Cantor en [Can03]. En un proyecto de software se pueden identificar tanto:

- **Tareas:** actividades, plantillas a llenar, operaciones, artefactos y prototipos que construir.
- **Entidades:** recursos humanos, tecnológicos, combinaciones de ambos que las ejecutan las cuales pueden ser consideradas como entidades ejecutoras de acuerdo a los planteamientos básicos de Booch, Rumbaugh y Jacobson [BRI00].

De la misma manera, estas tareas presentan una serie de dependencias entre ellas en la medida que la realización de algunas obedece a la ejecución previa de otras; y los recursos encargados de procesarlas tienen varios roles y por ende, responden de modo distinto al querer elaborarlas. La justificación de la aplicación de un algoritmo meta heurístico en la planificación de proyectos de Ingeniería de Software (tareas dependientes que pueden ser asignadas a recursos que tienen distintos rendimientos) es la siguiente:

- Las metodologías de desarrollo de software exponen las etapas por las que debe discurrir el proceso, pero no indican la forma de cómo hacer la clasificación y ordenamiento de las tareas involucradas (tareas, operaciones, objetivos, etc.) Esto toma especial importancia en las primeras etapas en donde se sistematiza todo el trabajo a realizar y donde se releva la información pertinente para iniciar el proyecto (requerimientos iniciales del cliente). La denominación de máquinas aquí puede cambiarse por el concepto de recursos involucrados en el proyecto: analistas, programadores, testadores, equipos, etc.
- Los métodos exactos planteados hasta el momento no pueden resolver instancias del problema de gran porte (ni para actividades de desarrollo de software ni para programación de tareas en líneas de producción) y menos aún, con estas características de dependencia de las tareas junto con la heterogeneidad en cuanto a las entidades que las van a ejecutarlas, como es el específico caso de los proyectos de

software.

- El proceso de analizar exhaustivamente todos los elementos de las mencionadas instancias (combinaciones) resulta muy costoso computacionalmente y en la mayoría de los casos, los métodos exactos pretenden hacer este tipo de búsquedas. Los algoritmos meta heurísticos GRASP, particularmente analizan un limitado grupo de elementos del espacio de candidatos a solución del problema y sobre ellos, aplica operaciones de combinación y optimización logrando soluciones de muy buena calidad.

1.3. La Administración de Proyectos y la planificación de recursos

También se puede encontrar relación directa del problema de la planificación de recursos o la planificación de tareas (conocido en la literatura como task scheduling) en la Administración y Desarrollo de Proyectos en general y en particular en los proyectos de Ingeniería de Software, como puede constatare:

- Las operaciones involucradas en un proceso de desarrollo de software presentan dependencia en tanto en cuanto forman parte de determinadas etapas.
- La Administración de proyectos debe procurar el uso adecuado y aprovechamiento de recursos (humanos, tecnológicos y temporales) buscando su máximo aprovechamiento lo que convierte claramente al problema en uno de Optimización Combinatoria.
- Las metodologías existentes indican los pasos *que se deben seguir* (o como se ha denominado *buenas prácticas*) para llevar con éxito un proyecto. En el tema referente a la organización de las tareas involucradas - sobre todo en las etapas iniciales en donde es fundamental la calidad de la información levantada (requerimientos), tampoco mencionan *cómo hacerlo* y podemos observar de inmediato la dependencia que ejercen estas tareas iniciales con la totalidad restante del proyecto.

La complejidad del task scheduling aplicado a la Administración de proyectos de Ingeniería de Software radica en estas características:

- Por un lado las tareas tienen predecesoras o sea, para su ejecución deben esperar previamente que se complete la ejecución de todas aquellas tareas la preceden: aparición del concepto tiempo muerto [Ste81].
- Las entidades que han de efectuar estas tareas no las hacen en el mismo tiempo: se presentarán restricciones en cuanto al rendimiento de cada una. Por ejemplo:
 - Si los ejecutores son recursos humanos, éstos tienen un horario de trabajo. Tampoco se pueden sobrecargar de tareas porque pueden ser de distinta naturaleza: analistas, programadores, documentadores, testadores, etc., situación

común en proyectos de desarrollo de software.

- Si se considera también a los equipos como recursos, éstos consumen energía y probablemente no pueden estar permanentemente activos (tal y como sucede en una línea de producción convencional).

Es por estas razones que dentro de las variantes del task scheduling, la variante que se desarrolla en el presente proyecto de tesis es la de mayor complejidad algorítmica siendo inviable el planteamiento de un algoritmo que la resuelva de forma exacta. Queda mostrar el panorama de una de las metodologías de desarrollo de proyectos de software más comunes y notar la importancia de la programación de tareas para formar el escenario donde sería posible la aplicación del algoritmo que se presentará en capítulos posteriores.

1.4. Aportes propuestos

Precisamente para tratar de llenar las carencias de algunas de las metodologías de desarrollo de software es que se plantea un algoritmo GRASP en la presente tesis: los métodos exactos con los que se ha pretendido resolver el problema (PERT; GANTT, etc.) a través de software comercial convencional (MS Project, PRIMAVERA Project, etc.) no contemplan características tales como dependencia de tareas y limitantes de recursos en simultáneo.

Se plantearán dos tipos de algoritmos:

- Un algoritmo voraz
- Un algoritmo meta heurístico GRASP con doble criterio de relajación (principal aporte y modificación al algoritmo original, tal cual lo plantearon sus autores).

Ambos serán enfrentados a un modelo matemático que intente arrojar las soluciones exactas para instancias más pequeñas.

1.5. Metodología sobre la que se aplican los algoritmos propuestos

La metodología seleccionada para verificar la situación de la programación de tareas involucradas en el proceso de desarrollo es RUP de la cual se esgrimirán conceptos básicos como los siguientes:

- Características y objetivos de la metodología
- Flujos de trabajo

- Disciplinas
 - Modelamiento de negocio: planificación de las tareas involucradas
 - Relevamiento de información: asignación de responsabilidades a los recursos disponibles
- Aplicación práctica de RUP en la administración de proyectos de software.

1.6. La planificación como antecedente del task scheduling.

La Optimización Combinatoria [Pa82] comprende una familia de problemas en donde las soluciones se obtienen a partir de procesos de ordenamiento, clasificación y combinación de los elementos que conforman el dominio del escenario a resolver. Una familia de esos problemas son los llamados problemas de asignación de recursos o de secuencialización de procesos de trabajo. Se observa que su aplicación encaja dentro de los escenarios de la programación de proyectos en general, incluyendo los de desarrollo de software e Ingeniería de Sistemas [Cha03].

El problema de la planificación consiste en asignar un orden de ejecución adecuado a un conjunto de operaciones que se han de efectuar siguiendo una línea de procesos o producción. Una de sus distintas variantes es conocida en la literatura como **task scheduling**, variante que se puede definir como sigue:

- Se tienen N tareas por procesar (es decir: actividades, plantillas a llenar, operaciones, artefactos y prototipos); cada una de ellas presenta una duración T unidades de tiempo.
- Se tienen M máquinas (en el caso que se desarrolla en la presente tesis estaría representado por recursos).

El objetivo es programar las N tareas en las M máquinas procurando el orden de ejecución más apropiado.

1.7. La Meta Heurística GRASP

Las meta heurísticas son métodos exploratorios [FR95] que permiten evaluar la calidad de los elementos que forman el escenario de un problema e ir formando la solución sin necesidad de escoger aquellos con mejor valor de la función de mérito, como sus pares los algoritmos voraces: relajan este criterio estableciendo listas de elementos candidatos a formar parte definitiva del conjunto solución, las cuales cumplen una serie

de restricciones.

De dicho conjunto de candidatas escogerá una al azar. Se ha demostrado que una aplicación sucesiva de las fases de construcción y mejoría del algoritmo generan mejores soluciones para problemas de Optimización Combinatoria.

1.8. Organización y estructura de la tesis

Se ha dividido el documento en los siguientes capítulos:

- Capítulo 1 - Introducción.
- Capítulo 2 - Marco teórico del problema: El problema de la programación de tareas o recursos: en este capítulo se busca pormenorizar la definición del problema así como sus principales características y métodos existentes para su solución, sea de forma exacta o aproximada. Se incluyen la justificación del uso de la metodología RUP, el análisis de su comportamiento particular y el planteamiento (planificación) de las tareas inmersas en sus primeras etapas; Se presentan además metodologías de planificación y aplicaciones adicionales existentes tanto para Ingeniería de Software como de índole industrial. Finalmente, se incluyen los fundamentos teóricos de los algoritmos GRASP.
- Capítulo 3 - Un modelo matemático de optimización para el problema de la planificación de recursos en los proyectos de Ingeniería de Software: se presenta el modelo matemático de esta variante del problema, los elementos del modelo y su importancia.
- Capítulo 4 - Algoritmos propuestos: en este capítulo se presenta la contribución realizada sobre la base de dos algoritmos: un algoritmo voraz y un algoritmo GRASP.
- Capítulo 5 - Implementación: en este capítulo se describe la forma de implementación de un prototipo que aplica los algoritmos propuestos.
- Capítulo 6 - Experimentos numéricos: en este capítulo se explican las pruebas a las que fueron sometidos los algoritmos; se presentan los resultados contrastados del algoritmo voraz frente al algoritmo GRASP para determinadas instancias de prueba, así como los resultados exactos para otras instancias más pequeñas y su comparación con los resultados arrojados por los algoritmos propuestos.
- Capítulo 7 – Análisis de los experimentos: en este capítulo se hace un análisis de exhaustivo sobre el rendimiento de los algoritmos frente a varios factores (iteraciones, valores de las constantes de relajación, supuestas soluciones exactas).
- Capítulo 8 – Conclusiones: Aquí se darán las conclusiones del proyecto, potenciales trabajos futuros y recomendaciones finales.

Capítulo II Marco teórico

2.1. La planificación como problema de optimización y antecedente del task scheduling.

La Optimización Combinatoria [Pa82] comprende una familia de problemas comunes con ligeras variaciones entre sí, entre los que encontramos a los problemas de asignación de recursos o de secuencialización de procesos de trabajo. Se ve que su aplicación encaja dentro de los escenarios de la programación de proyectos en general, incluyendo los de desarrollo de software e Ingeniería de Sistemas [Cha03].

El problema de la planificación consiste en asignar un orden de ejecución adecuado a un conjunto de operaciones que se han de efectuar siguiendo una línea de procesos o producción. Dentro de las distintas variantes que presenta la planificación encontramos a la **programación de tareas** conocido en la literatura como **task scheduling**, variante que se puede definir como sigue:

- Se tienen N tareas por procesar (es decir: actividades, plantillas a llenar, operaciones, artefactos y prototipos); cada una de ellas presenta una duración T unidades de tiempo.
- Se tienen M máquinas (en el caso que se desarrolla en la presente tesis estaría

representado por recursos).

- El objetivo es programar las N tareas en las M máquinas procurando el orden de ejecución más apropiado.

La complejidad del task scheduling va en aumento a medida que se van estableciendo restricciones adicionales referidas a la naturaleza de las máquinas y las tareas involucradas: por ejemplo dependencia entre tareas (es decir, cada tarea del proyecto de Software presenta una lista de tareas que la preceden) y heterogeneidad de las máquinas es decir de los trabajadores (una misma tarea puede ser ejecutada en tiempos distintos, por diferentes trabajadores).

Esto acontece directamente, como se indicó en la justificación, en la planificación de proyectos de software: las operaciones a realizarse suelen mostrar dependencia mutua mientras que los recursos ejecutores (humanos y tecnológicos) presentan una serie de restricciones tanto temporales como de rendimiento: horarios de trabajo, cantidad de tareas concurrentes que pueden ejecutar, límites de tiempo en los cronogramas, disponibilidad de los equipos, que los convierten en heterogéneos. Y también en dichas planificaciones se ha de cumplir con ciertos cronogramas-los llamados deadlines- con lo que se busca minimizar el tiempo total de trabajo junto con mejorar el uso del resto de recursos.

2.2. Estado del arte del problema de la Planificación de Tareas

La planificación consiste en asignar un orden de ejecución adecuado a un conjunto de operaciones que han de ejecutarse concurrentemente. Un plan viene a ser un proceso jerárquico a través del cual un organismo de producción (por ejemplo un equipo de proyecto dentro de una Consultora de Sistemas, etc.) puede controlar el orden en el que una secuencia de operaciones es ejecutada en su interior [MG60]. Obsérvese que no contraviene esta definición en modo alguno, a la definición que pueda presentar el plan de actividades implícitas en un proyecto de Desarrollo de Software.

Usualmente, siempre que la Matemática Computacional o la IA han desarrollado formas de resolver el problema de la planificación sí se han enfocado en procesos industriales particulares y en el análisis de la naturaleza de los elementos involucrados: operaciones, maquinarias y comportamiento de la línea de producción.

Para el proceso de planificación existen tres elementos fundamentales:

- Tareas, trabajos u operaciones a ejecutar (ordenar).
- Recursos ejecutores de dichas tareas: en el caso de un proyecto de software, los diferentes actores (roles) interventores en el proyecto.
- Un algoritmo planificador – ordenador – administrador de recursos.

En la práctica (incluyendo la Administración de proyectos de Desarrollo de Sistemas) se buscan ordenar los trabajos a realizar, tentando reducir u optimizar el funcionamiento de las entidades que los ejecutan (menos tiempo ocioso o menor tiempo total de procesamiento, mejor empleo de los recursos, etc.) para tratar de producir un resultado satisfactorio dentro de ciertos parámetros.

En la misma línea, cuando se calendarizan los proyectos de Desarrollo De Software – tareas de definición, tareas de desarrollo y tareas de soporte – también se empezaron a usar métodos de Investigación de Operaciones para dar una secuencia a dichas fases [Zah94], aplicando algunos métodos de recorridos y determinación de rutas críticas como son Pert [Kir66], Gantt, etc.

Así, el proceso de planificación de tareas implica un uso adecuado de los recursos con el fin de poder garantizar ciertos requisitos temporales y de optimización. Un método o estrategia de planificación debe presentar dos aspectos fundamentales:

- Un algoritmo de planificación que determine el orden de acceso de las tareas a los recursos de la línea: procesadores o máquinas; células de trabajo.
- Un método de análisis del plan resultante que permita calcular el comportamiento temporal de la línea o mundo. Así se puede comprobar si los requisitos temporales se están garantizados en todos los casos posibles [Si99].

Las variantes que pueden presentar al problema aparecen por las particularidades que presenten tanto de las tareas como de los ejecutores involucrados, así como de lo que se considere el objetivo a optimizar; de la misma manera, la disposición con que son hechas (sea en línea o flujo, con presencia de colas de espera, con tiempo de penalidad, paso de cada tarea por todas las máquinas de forma obligada, etc.) generarán otros tipos de problemas muy conocidos y profundamente estudiados en la literatura(referencias).

2.2.1. Objetivos del scheduling

En la Industria del Software encontramos como factor fundamental el cumplimiento de metas dentro de determinados plazos de tiempo: el recurso tiempo, es quizá el más caro y probablemente el más complejo de manejar por su carácter de irrecuperable [Pro00]. Para este fin se busca mantener activos a los recursos involucrados en el proyecto (sean éstos recursos humanos o no) el mayor tiempo posible, dentro de las propias restricciones que presente. Esto debido a que los recursos también constituyen gastos, si es que permanecen constantemente ociosos. Las soluciones existentes para el scheduling tratan de minimizar en lo posible el tiempo ocioso de las máquinas en nuestro caso trabajadores involucrados en el proceso y el tiempo de proceso total del conjunto de trabajos a realizar en el proyecto, conocido en la literatura como makespan.

2.2.2. Variantes del problema del scheduling

Como se mencionó antes, dependiendo de las propiedades de los componentes del scheduling es que surgen sus variantes particulares.

- Job scheduling [BF89]: un lote o conjunto finito de trabajos divididos en operaciones que serán procesados o ejecutados en un determinado número de procesadores ¹ (máquinas-entidades ejecutoras).
- Task scheduling: Se puede establecer las siguientes subvariantes:
 - Máquinas idénticas y tareas independientes [Ku88]
 - Máquinas idénticas y tareas dependientes [Ste81]
 - Máquinas diferentes y tareas independientes [ECP97] [Tu01].
 - Máquinas diferentes y tareas dependientes: el modelo más complejo que será motivo de estudio en este trabajo.
- Flow-shop scheduling [Co67]: las tareas tienen que pasar por todas las máquinas, una a la vez (estén o no divididas en operaciones independientes). Se consideran que las máquinas son idénticas y su disposición es de forma paralela.
- Programación en tiempo real

Se encuentra una adaptación lógica para los problemas de un proyecto de software dado que las tareas presentan dependencias (en relación a todo el proyecto y dentro de su respectiva fase RUP) y que los recursos ejecutores disponibles son diferentes por los roles que ejecutan. Esta es parte de la justificación del proyecto como se ha indicado en el capítulo 1. A manera de resumen, la siguiente ilustración muestra las principales divisiones y variantes presentadas en un esquema que parte de la planificación y termina con las variantes particulares del task scheduling, que son motivo de estudio de la presente tesis:

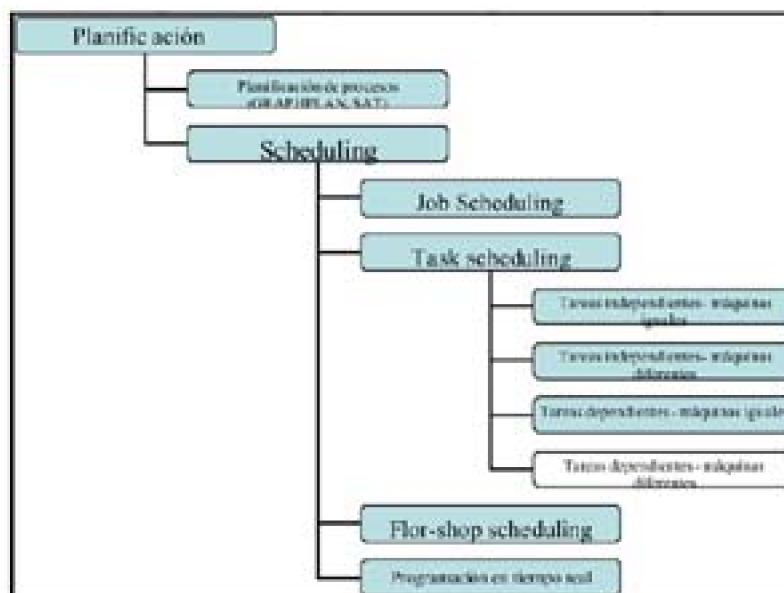


Figura 2□1: Algunas variantes de la planificación industrial.

¹ : El término procesador es usado por que las primeras aplicaciones de algoritmos planificadores fueron hechas para programar De la misma forma, la siguiente tabla muestra de forma resumida, las diferencias tareas en los procesadores de la naciente micro electrónica a mediados de la década de los años 50, en el siglo XX (N. del A.)

existentes entre las variantes del scheduling presentadas en este apartado, destacando la variante del task scheduling:

Variantes Principales	Características			
	Las tareas se ejecutan en paralelo	Contempla dependencia entre las tareas	Se tiene que pasar por todas las máquinas	Hay procesamiento en paralelo
Job scheduling	Si	No	Puede	Si
Task scheduling	No	Si	No	No
Flow-shop scheduling	Puede	No	Si	No
Programación en tiempo real	No	Puede	Puede	Si

Tabla 2□1: Características resumidas de cada una de las variantes del scheduling.

2.3. Análisis de la variante del task scheduling propuesta

Esta variante es la de mayor dificultad de todas las vistas debido al comportamiento de cola que tienen las tareas y la heterogeneidad de las máquinas. Podemos presentar una nueva definición como sigue:

- Se tienen M recursos ejecutores con diferentes rendimientos.
- Cada tarea se ejecuta en un tiempo t_j que es el tiempo que demora la tarea i-ésima en ser ejecutada por el recurso j-ésimo.
- Se tiene una lista de predecesoras por cada tarea.
- Se tiene una matriz de tiempos de ejecución de cada tarea.

2.4. Métodos existentes para resolver el problema del scheduling.

Para la resolución del problema del scheduling se plantean dos tipos distintos de métodos:

- Métodos exactos: los que pretenden calcular una solución exacta.
- Métodos aproximados: entre ellos, las técnicas heurísticas y meta heurísticas

Los métodos exactos no resuelven en particular las variantes vistas en los apartados anteriores. Más bien, se enfocan a resolver modelos híbridos o planteamiento de la planificación industrial tradicional o de Investigación de Operaciones; por este motivo mencionaremos a métodos como GRAPHPLAN y STRIPS. Véase el esquema siguiente:

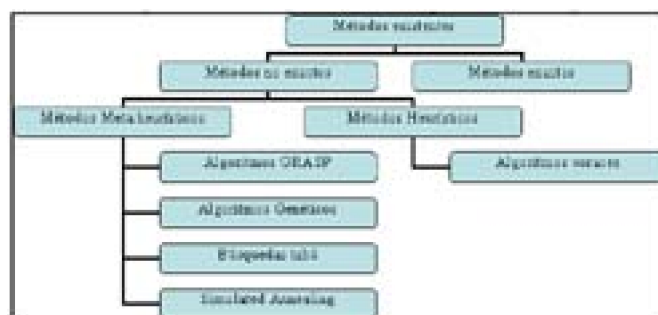


Figura 2□2: Métodos existentes: exactos y aproximados

Entre los métodos exactos se puede mencionar a:

- GRAPHPLAN [SVD80] [IJCAI95] [Lo99]
- Algunas variantes de SAT, PERT, CPM [Ka92]
- STRIPS [FN71] [We99] [Wi84]
- GPS: General Problem Solver [NS63]

Estos métodos constituyeron un gran avance para el área de la planificación a pesar de que son procedimientos genéricos de solución más no específicos para el problema del scheduling. La aplicabilidad de dichos métodos radicaba en poder transformar los distintos escenarios del scheduling de tal forma que encajen en las condiciones que requieren, para poder hallar la solución requerida. Este proceso era imposible para la mayoría de variantes más complejas. Se comprobó que, convirtiendo el problema del scheduling a la forma del GRAPHPLAN (grafos dirigidos), se podían obtener resultados relativamente buenos y a costos computacionales bajos. Sin embargo, las variantes del scheduling que se pueden resolver de esta forma son las más sencillas y los algoritmos heurísticos lo superan ampliamente.

2.4.1. Métodos heurísticos y meta heurísticos planteados por variante

Consideremos al scheduling como un conjunto de N tareas y de M máquinas, buscando como objetivo principal la **minimización del tiempo de procesamiento o makespan** [BGKRS95]. Las heurísticas en este tipo de problemas, buscan primero adoptar una estrategia de ordenamiento de tareas y / o máquinas en nuestro caso trabajadores, previo al proceso de asignación. Una evaluación de la calidad de la solución se realiza mediante la comparación con un límite inferior de duración (criterio voraz o GRASP), siempre y cuando el objetivo sea la minimización del tiempo (existen como hemos afirmado antes, otros objetivos).

Si bien a partir de este algoritmo o criterio heurístico se pueden generar algoritmos meta heurísticos (como los algoritmos GRASP, heurísticas Tabú [GKL95], algoritmos genéticos), en la actualidad muchas de las soluciones encontradas sólo se centran en un criterio voraz o voraz para plantear su metodología, por la calidad de las soluciones halladas. En la siguiente ilustración se presentan los tipos de métodos heurísticos y meta heurísticos existentes hasta el momento para resolver las diferentes variantes del

problema:

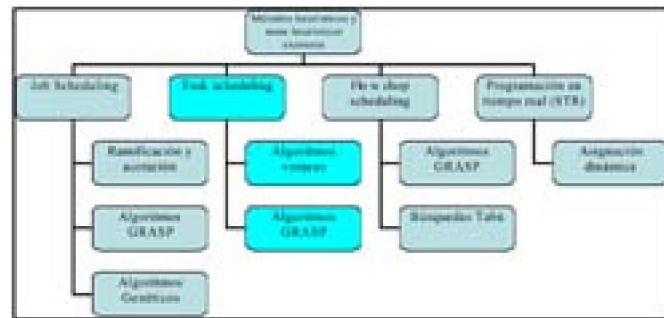


Figura 2□3: Métodos heurísticos para resolver las variantes del scheduling

En casi cualquier área del saber humano se puede ofrecer soluciones a problemas de planificación de acciones y diseño de planes de trabajo. Veremos algunos ejemplos en variadas ramas de la Tecnología de la Información (TI) en donde se han planteado soluciones exactas o aproximadas:

- Ingeniería de Software e Informática
 - Establecimiento de planes de desarrollo: buscando manejo y administración de costos [Cha03], [Ros05]
 - Administración y dirección de proyectos de Ingeniería de Software [Can01], [Can01b], [Char04].
 - Dirección y control de decisiones dentro de procesos de desarrollo de software [McI00].
 - Dirección y modelamiento de pequeños proyectos de rápido desarrollo [Pol01].
 - Planificación de procesos de alto nivel dentro del desarrollo de software [We02].
 - Estimaciones de planes para grandes proyectos [Kro04], [Roy05].
 - Aplicaciones de TI a los negocios en proyectos de desarrollo de software [Red04].
 - Programación de tareas de sistemas operativos [YWCH02]
- Gestión Empresarial: establecimiento de planes de proyectos de inversión [MIC01].
- Diseño automático de circuitos integrados – arquitectura de computadoras: el modelado de circuitos [HPL03], minimizaciones lógicas, asignaciones de estados, minimización de estados, síntesis de circuitos asíncronos, codificación de entradas y salidas para máquinas secuenciales, etc. Dentro de la optimización de conjuntos de instrucciones [ICM96], optimización de controles de computadoras, diseño de circuitos lógicos, optimización de compiladores, etc.
- Control de operaciones: manejo y planificación de operaciones en plantas industriales [Ja03].
- Logística de transportes: establecimiento de planes para transporte [Ri95].
- Industriales: establecimiento de planes de trabajo para líneas de producción. Aquí

encontramos también variadas aplicaciones específicas:

- Planificación de procesos: referido tanto a la distribución temporal de la gestión de un pedido de producción como a la asignación de recursos en el proceso de fabricación en sí [BF89].
- Programación de líneas de producción: buscando una minimización del tiempo total de trabajo, el cumplimiento de un tiempo límite máximo, la reducción de tiempos muertos u ociosos; minimización de costos en el uso de insumos; multiobjetivos (combinaciones de varios objetivos)
- Programación de máquinas y robots: planificación de las acciones que realizan los autómatas con algoritmos de control numérico [Mom87], buscando la modernización de los sistemas CAD y CAM [ICRA99]
- Industrias de la manipulación: minimización de desplazamiento (de autómatas ensambladores) en la transformación de piezas [Oco84].
- Industrias del ensamblado: planificación del ensamblado de piezas grandes (autos, computadoras, componente, maquinaria pesada, etc.) [BF95], buscando la minimización de tiempos de producción y de defectos [CAIA90].

2.5. Metodología sobre la cual se aplicarán los algoritmos: RUP.

RUP [BRI00] es un proceso de desarrollo de software, de Ingeniería de software. Es un conjunto de actividades que permite transformar los requisitos de un usuario a un sistema de información que resuelva algún problema que éste tenga:



Figura 2□4: Modelo general del task scheduling

Proporciona lineamientos específicos para llevar a cabo determinadas fases del proceso [Kru00] desde el modelamiento de negocios, diseño, arquitecturas, implementación hasta las pruebas y el continuo proceso de control de la calidad. Se presenta así, como una forma disciplinada de asignar tareas y responsabilidades en una empresa o equipo de desarrollo (quién hace qué, cuándo y cómo). A continuación se señalan algunos de los objetivos perseguidos y las características fundamentales de la metodología:

- Objetivos básicos:
 - Asegurar la producción de software de calidad dentro de plazos y presupuestos predecibles.

· Características

- Utilización de UML para el proceso de modelamiento planteado por Booch, Rumbaugh y Jacobson en [BRI99].
- Es dirigido por casos de uso.
- Está centrado en la arquitectura.
- Es de carácter iterativo (mini-proyectos progresivos) e incremental (versiones en aumento)

Estas últimas características son los aspectos definitorios de RUP. Dada su importancia en la planificación de todas las etapas involucradas en el proceso de desarrollo, se analizarán brevemente y por separado.

2.5.1. Característica 1: RUP está dirigido por medio de casos de uso (use case)

Un modelo es una representación de una situación o conjunto de estados de ciertos aspectos de la realidad. Capta aspectos importantes del escenario en el que se está inmerso dentro de un proyecto, desde determinados puntos de vista: el modelo de un sistema de software esta construido en un lenguaje de modelado como UML con el objetivo de obtener los requisitos de dicho sistema dentro del dominio del problema que pretende resolver [Boo90].

La interacción entre un usuario y el sistema se puede definir, igualmente, como un caso de uso: aspecto de la funcionalidad del sistema que le permite obtener al usuario algún tipo de resultado útil y valedero. Como lo mencionan Booch, Jacobson y Rumbaugh [BRI99] responden aproximadamente a la pregunta ¿qué va a hacer el sistema al relacionarse con el usuario que lo maneje, para cada tipo de usuario existente? Los casos de uso guían así al proceso de desarrollo en la medida que muestran cómo diseñar, implementar y probar el sistema.

A partir de ellos se va conduciendo el resto de etapas por las que pasará RUP: seguir el hilo a través de una serie de flujos de trabajo [Schn01]. Es ahí donde se hace necesario una muy buena organización y planificación de las tareas para que este hilo conductor no lleve a situaciones erróneas. Se justifica entonces, la necesidad de ordenar las operaciones no sólo como requisitos funcionales para la construcción de casos de uso correctos, sino también para desenvolver adecuadamente el resto del proceso.

2.5.2. Característica 2: RUP está centrado en la arquitectura

La arquitectura de un sistema [Can03b] incluye sus aspectos estáticos y dinámicos más significativos: entiéndase por aspectos desde las necesidades del usuario, las políticas que se aplicarán al sistema de parte de quienes lo van a usar, los presupuestos, las

plataformas de funcionamiento (ejemplo sistema operativo donde se instalará), requisitos no funcionales como fiabilidad, velocidad de respuesta, etc.

Ahora se puede establecer una relación entre función y forma: los casos de uso se deben poder ejecutar en el escenario denominado arquitectura; ésta debe permitir además, la realización adecuada de las funciones. La arquitectura es pues la forma del sistema basada en las funciones que va a cumplir: también aquí vemos que con una comprensión de los casos de uso y un progresivo refinamiento de lo que hace el sistema, se logra el desarrollo adecuado de RUP [Hum00].

2.5.3. Característica 3: RUP es iterativo e incremental

Una característica fundamental que propugna RUP es la división de todo el proyecto de software en partes más pequeñas. Cada mini proyecto es una iteración que implica un incremento en el avance del proyecto, así:

- Una iteración es definida como una etapa en la que se han realizado concurrentemente, ciertas operaciones del proceso.
- Un incremento significa la modificación del producto (va creciendo).

Los casos de uso más relevantes en una determinada etapa son identificados, son implementados dentro de la arquitectura más adecuada y luego se realiza la verificación sobre si los componentes diseñados los satisfacen. La cantidad de iteraciones y las acciones a realizarse en cada uno de ellos tienen que ser planificadas de manera muy precisa: *podemos observar temas como dependencias y restricciones para pasar de una iteración a otra y empezamos a deslizar la idea de programación de estas tareas, junto con los responsables (máquinas a partir de la definición básica del problema del scheduling) de ejecutarlas.*

A medida que se van realizando las iteraciones, sea hacia delante o volviendo a corregir errores de anteriores, se van teniendo prototipos de las distintas versiones por las que se va pasando; de manera que cada iteración arroja un producto cada vez con mayores funcionalidades (más y mejor) que el anterior [Co02].

2.5.4. Otra característica: RUP también es un producto

Desarrollado y mantenido por Rational, RUP ofrece una serie de productos y ambientes con los que se puede administrar las distintas etapas de los procesos y fases por las que pasa. Permite el acceso simultáneo a bases de conocimiento, plantillas y herramientas, centrándose en la producción y mantenimiento de modelos del sistema más que en producir documentos. Existen herramientas RUP de apoyo a todo el proceso, brindado soporte a todas las características indicadas:

- Desarrollo iterativo del software
- Administración de requerimientos

- Uso de arquitecturas basadas en componentes
- Modelamiento visual del software
- Verificación de la calidad del software
- Control de cambios

Los casos de uso y los escenarios indicados donde ocurren, por el proceso han probado ser una buena forma de captar requerimientos y guiar el diseño, la implementación y las pruebas.

2.5.5. Los recursos convertidos en *máquinas*

Un sistema – o mejor dicho, su desarrollo, tiene varios elementos involucrados que comúnmente son denominados recursos: humanos, tecnológicos, económicos, temporales, etc. El término trabajador designa a la persona que realiza determinada actividad dentro del proyecto, pudiendo ser de distinto tipo como se ha estado indicando.

Para la adaptación adecuada de la propuesta del presente proyecto de tesis, se ha creído conveniente relacionar el término **trabajador** con la acepción **máquina** de modo que se entienda que es el trabajador el que ejecuta la tarea, así como una máquina (o célula de trabajo obrero-herramienta) puede ejecutar las operaciones dentro de una línea de producción.

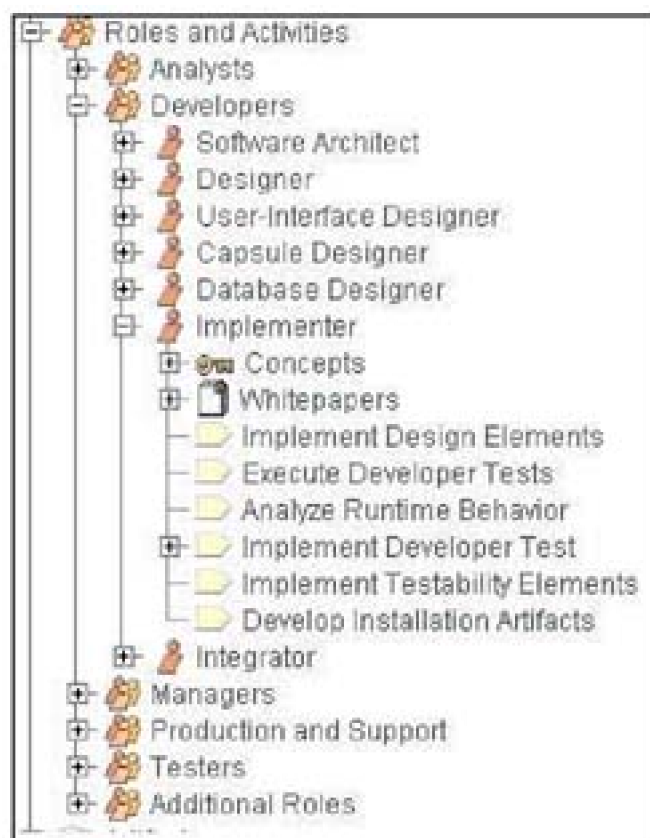


Figura 2□5: Algunos roles en RUP [Pol04]

2.5.6. Las fases de RUP

RUP divide el proceso de desarrollo en ciclos, teniendo un producto al final de cada uno denominado **versión** del producto. Por lo mismo, se requiere de una planificación adecuada de las tareas involucradas dentro de un ciclo; cada uno a su vez, se divide en cuatro fases:

- Inicio: también denominado fase de comienzo o concepción.
- Elaboración
- Construcción
- Transición

Cada fase concluye con un hito bien definido donde deben tomarse ciertas decisiones, lo que demuestra la dependencia de las tareas entre ellas. Véase la siguiente figura:

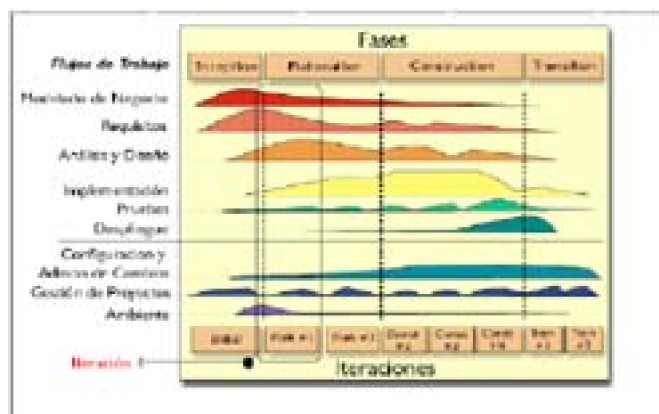


Figura 2□6: Fases de un ciclo RUP

Lo que está planteando RUP es un cuadro de doble entrada donde en cada interacción (en la parte inferior – sentido horizontal) se desarrollan en mayor o menor grado, las etapas de los flujos de trabajo (parte izquierda – sentido vertical). Una fase puede contener una o varias iteraciones. Vamos a hacer incidencia más adelante, en las fases iniciales y en los flujos de trabajo superiores para comprobar la importancia de una organización adecuada de las tareas que se han de efectuar.

2.6. Aplicación práctica de RUP en la planificación de proyectos

El planeamiento de actividades dentro de un proyecto de software es una actividad difícil, que exige creatividad precisamente para establecer un equilibrio entre las circunstancias que afectan al proyecto – en realidad, las que afectan al escenario del

cliente que necesita el software, a los objetivos a cumplir, las restricciones, etc. [Lo03]. Se tiene una fuerte interdependencia entre las acciones que se van a ejecutar en todo el desarrollo, por lo que el establecimiento de un plan de proyecto no es una labor tan trivial como limitarse a designar *¿quién va hacer qué?*

El proceso de construcción del plan consiste en ir haciendo esbozos que vayan cubriendo de poco a poco, la mayor cantidad posible de restricciones comenzando de las más importantes (críticas para el proyecto): a este refinamiento consecutivo es al que se refiere RUP en los planteos que se presentaron en los apartados anteriores. Muchas teorías sobre el ordenamiento y planeamiento de operaciones durante los proyectos de desarrollo de software presentan recetas conocidas como mejores prácticas pero no consideran muchos aspectos de la realidad (el *¿cómo?*) [Kr02][Ro98]: básicamente se debe seguir los pasos que fueron correctos para otros proyectos y adaptarlos en la medida que sean necesarios.

Así no se está considerando tres factores fundamentales que presenta por ejemplo, López [Lo03] en su análisis del comportamiento de un proyecto de software: administración de la planificación en sí, la estimación y la negociación. Variables que el autor indica, siempre son dejadas de lado por los administradores o jefes de proyecto cuando trazan el plan de operaciones. Todos son pasos previos, dependientes entre sí y que deben seguir un orden preciso según sea cada caso (cada desarrollo) en particular.

RUP plantea la construcción de planes de forma iterativa partiendo de ciertas actividades generales básicas:

- Definir los objetivos del proyecto
- Definir las fases
 - Calendarizar el proyecto en cuanto al esfuerzo estimado a realizarse por fase y al paso de una a otra.
 - Priorizar riesgos: manejo de las restricciones de tiempo y de presupuesto sobre todo.
 - Definir hitos: como puntos de control de avance (gestión del cambio incluida)
 - Definir iteraciones
- Calendarizar nuevamente el proyecto en cuanto al esfuerzo estimado a realizarse por iteración y al paso de una a otra.

Cada etapa trae consigo una serie de operaciones asociadas, y ahí radica el vacío de las metodologías de planeamiento de proyectos: no indican *cómo hacer el ordenamiento de las citadas* operaciones. Es más, RUP aclara que este orden de operaciones básicas es modificable (o adaptable) y no único.

2.6.1. Programando los objetivos definidos del proyecto RUP

De por sí la definición misma de los objetivos es una tarea harto compleja como lo indica

Sommerville en [Som92]: los objetivos indican lo que se va a hacer en el proyecto y muestran qué parte del problema de negocios del cliente va a ser afectada (positivamente, pues se va a resolver un problema) pero no nos dice porqué ni cómo.

Justamente para definirlos se requiere conocimientos sobre el contexto en que se va desenvolver todo el desarrollo y los recursos involucrados; pero, a pesar de que no mencionan cómo lograrlos, deben venir acompañados de actividades (comúnmente llamados criterios de obtención) generales para lograrlos ². Los grandes objetivos pueden descomponerse en una serie de tareas relacionadas entre sí, aplicando luego la propuesta algorítmica de la presente tesis para definir el orden adecuado de ejecución. Para esto adicionalmente se necesita de una descripción adecuada de las fases del proyecto y de lo que se pretende desarrollar en cada una de ellas, cosa que RUP presenta.

2.6.2. Empleo de las fases RUP y su relación con la programación de tareas

Nuevamente RUP nos indica lo que se debe hacer en cada una de sus fases como parte del ciclo de vida por el que pasa el desarrollo. A continuación, algunas de esas indicaciones ³:

Tabla 2□2: Las tres primeras fases RUP y sus objetivos involucrados

² : Estas actividades pueden ser plasmadas en el documento de visión del proyecto según lo plantea PMI [PMIBOOK00] (N. del A.)

³ : Vendrían a ser los objetivos de las fases [We02] (N. del A.)

Fase	Objetivos involucrados
Concepción	Obtener un enfoque general del escenario donde se desarrolla el proyecto
	Hacer un análisis económico real de todo el proyecto
	Lograr un entendimiento mutuo (cliente - desarrolladores) de las funciones críticas del sistema a construir
	Realizar una estimación real del esfuerzo, el costo, el uso de recursos y un plan de trabajo preliminar
	Reconocer los riesgos implicados en el proyecto
Elaboración	Demostrar que la arquitectura del sistema va a satisfacer los requerimientos y limitaciones impuestas por el producto
	Listar las actividades que garanticen la estabilidad de la visión final del producto
	Elaborar un segundo plan de trabajo (más realista y ajustado que el primero)
	Eliminar algún riesgo significativo que atente contra el éxito del proyecto
Construcción	Construir y probar el sistema de forma incremental
	Promover el entendimiento y la aceptación del sistema por parte de los usuarios
	Minimizar costos y ajustar el plan de trabajo

Nótese que las tareas dadas siguen siendo muy generales, aunque sí son diferenciadas por cada fase y van mostrando que a medida se va avanzando en el ciclo de vida, hay muchas más operaciones involucradas y subordinadas entre sí.

2.6.3. Calendarizar el proyecto en cuanto al esfuerzo estimado a realizarse por fase

De manera progresiva, RUP indica que el esfuerzo estimado (qué tan involucrado están los recursos del proyecto – o máquinas para nuestro propósito) va en aumento con el correr del desarrollo, y nos provee de formas de calcularlo así como porcentajes de ocupación de recursos por fase. Es obvio que para la aplicación de estas medidas, es necesario tener previamente las tareas del plan global lo suficientemente ordenadas y allí es donde radica la importancia de un algoritmo planificador como el que se propone en esta tesis.

2.6.4. Priorizar riesgos y manejo de restricciones

Una gradual eliminación de los riesgos es una práctica fundamental en la Administración moderna de proyectos, cualesquiera sea su naturaleza. Como parte de la metodología de RUP (quien brinda como producto, herramientas para la administración del riesgo) se sugiere identificar esta lista de riesgos, analizarlos y ordenarlos de acuerdo a su impacto por lo que también se necesita una técnica de ordenamiento ⁴.

⁴ : RUP considera los siguientes factores para establecer una jerarquía de riesgos: impacto, severidad y prioridad (N. del A.)

2.6.5. Definición de hitos y de iteraciones dentro de cada fase

Como resultado de la definición de los objetivos, riesgos y planes preliminares para cada una de las fases se deben establecer hitos que se comporten como productos finales de cada iteración y que indiquen el estado del desarrollo del producto. Un hito puede mostrar la correlación guardada por las actividades realizadas dentro de cada fase finalizada y se presentan como fuentes de decisión para las fases por venir. De la misma manera, la definición de las fases RUP trae consigo la definición de los flujos de trabajo dentro de cada una de ellas (disciplinas) y las iteraciones que se van a llevar a cabo junto con las actividades para lograrlas. RUP sugiere un número promedio de iteraciones por fase, para los proyectos pequeños – medianos:

- 1 iteración para la fase de Concepción
- 2 iteraciones para la fase de Elaboración
- 2 iteraciones para la fase de Construcción
- 1 iteración para la fase de Transición

Aquí lo que se pretende es ordenar determinadas tareas (determinados objetivos en realidad) para cada iteración, eliminando de así los riesgos más importantes. De nuevo, se precisa de una especificación del orden de esas operaciones para garantizar el cumplimiento de los objetivos de cada fase.

Hasta este punto se puede concluir que en realidad, en un proyecto de Ingeniería de Software, hay que construir varias listas y hay que diseñar varios planes (en base a refinamientos continuos) cuando se aplica la metodología RUP. Pero también se ha detectado un vacío compartido por otras metodologías de desarrollo: esta claro lo que se debe hacer e incluso se desliza cierto orden pero no se indican las tareas para lograrlo. Al menos esto debería existir para las fases iniciales y para las primeras disciplinas: la propuesta que se realiza en esta tesis pasa por el hecho de organizar y ordenar cualquier lista de tareas concurrentes dentro del proyecto de Ingeniería de Software; sean éstas lista de tareas, objetivos y riesgos.

Otros temas importantes que también requieren de este tipo de criterios de planificación y ordenamiento son motivos de estudio por parte de RUP: el tema de las pruebas al software, los prototipos construidos, el control y administración de los cambios en el proyecto, etc.

2.6.6. Las dos fases iniciales RUP: el problema de planificar las operaciones involucradas

Se ha considerado que en las fases iniciales del desarrollo de un proyecto de Ingeniería de Software es indispensable poseer un flujo adecuado de eventos y una organización pertinente de las operaciones. En este apartado veremos brevemente en qué consisten las etapas y los flujos de trabajo que se suponen críticos.

2.6.6.1. Fase de Inicio o Concepción: tareas y objetivos involucrados

Esta primera etapa exige la definición de la idea general del proyecto y su posterior refinamiento de manera que sirva como rectora del restante desarrollo. Se establece la oportunidad y alcance del proyecto con una visión macro: o sea, lo que se pretende obtener en adelante, mas no la forma cómo lograrlo. Se identifican también, todas las entidades externas con las que se trata (actores) y se define la interacción a un alto nivel de abstracción por medio de:

- Identificación de todos los casos de uso y descripción de algunos en detalle.
- Identificación tanto de los criterios de éxito como de los riesgos
- Estimación inicial (que se debe refinar) de los recursos necesarios
- Plan de las fases incluyendo hitos.

Todos estos temas de alguna manera tienen que ser planteados de forma genérica desde esta etapa, por lo expuesto en apartados anteriores. El objetivo fundamental es obtener el modelado de negocios: *lo que pasa ahora y lo que se quiere resolver con el software que se pretende construir* [JEJ95]. En efecto dentro de los flujos de trabajo que analizaremos más adelante, la captura de requisitos y este modelamiento deben ser avanzados en gran medida (en detalle, minimizando riesgos) en esta fase. Son productos de la concepción entre otros, los siguientes:

- Un documento de visión general
- Determinación de los requerimientos generales del proyecto
- Determinación de las características principales (técnicas y no-técnicas)
- Determinación de las restricciones.
- Modelo inicial de casos de uso (10% a 20 % listos).
- Glosario de términos.
- Casos de uso de negocio, cuando amerite
- Identificación inicial de riesgos.
- Determinación (abstracta según algunos autores – sujeta a refinamientos posteriores [Can01]) de un plan de proyecto.

2.6.6.2. Fase de Elaboración

En esta etapa se busca principalmente, determinar la arquitectura del software. Por arquitectura se puede entender (como se menciona en apartados precedentes) al conjunto de ideas que permiten definir la organización de un sistema, sus componentes, las interfaces entre ellos y sus usuarios, su comportamiento, el conjunto de subsistemas y su funcionamiento, restricciones, funcionalidad, flexibilidad en la respuesta, tecnología usada, etc.

El grueso de las actividades que se realizan en esta etapa corresponde a las

disciplinas de captura de requisitos, análisis y diseño: todas ellas involucran una serie de operaciones con un marcado sesgo de dependencia. A manera de resumen véase la siguiente figura:

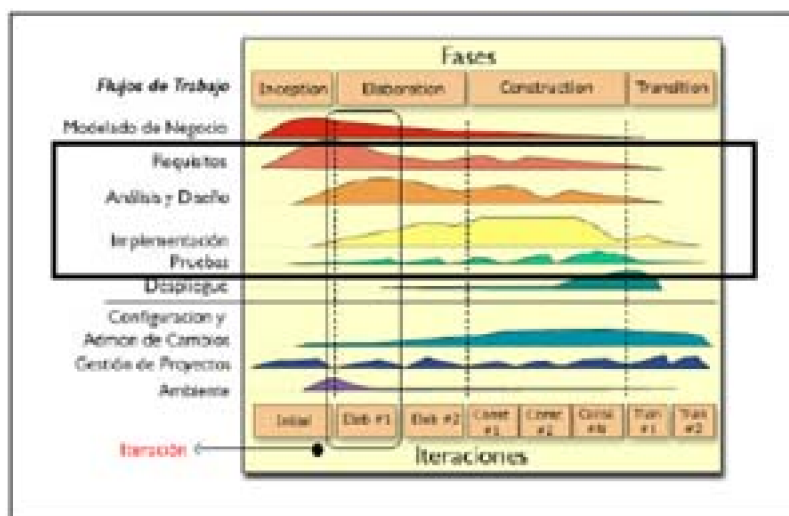


Figura 2□7: Fases de Concepción y Elaboración RUP y flujos de trabajo incluidos

Un estimado del esfuerzo de algunas disciplinas o flujos de trabajo por iteración simple, en cualquiera de fases RUP, es presentado en la siguiente tabla [Lo03]:

Tabla 2□3: Porcentaje de esfuerzo por flujo de trabajo en una iteración simple

Flujo de trabajo	Porcentaje de esfuerzo estimado promedio
Requerimientos	9%
Análisis y diseño	15%
Implementación	36%
Pruebas	21%
Administración	11%

Justamente en las dos fases referenciadas, el porcentaje de esfuerzo sufre una modificación pues el modelado de negocios, los requerimientos y el análisis y diseño son importantes en al actuar como punto de partida para las siguientes.

En este punto se puede distinguir otra vez la necesidad de la determinación adecuada de dichas actividades y la carencia existente en las metodologías para proceder a su ordenamiento. Queda puntualizar algunos conceptos sobre los flujos de trabajo y el papel que juega la planificación de tareas en ellos.

2.7. Flujos de trabajo o disciplinas: planificando recursos involucrados

RUP define a un flujo de trabajo o disciplina (discipline de la literatura en habla inglesa) como la realización práctica de un caso de uso de negocio ⁵ o parte de él. Reúne al conjunto de actividades hechas por el(los) trabajador(es) participante y los artefactos que producen: rápidamente podemos efectuar la analogía con una línea de producción industrial sin perder generalidad. Los flujos de trabajo fundamentales que aparecen por iteración son:

- Captura de requisitos
- Análisis
- Diseño
- Implementación
- Pruebas

En la siguiente figura se observa un diagrama de actividad dentro de un proyecto convencional:

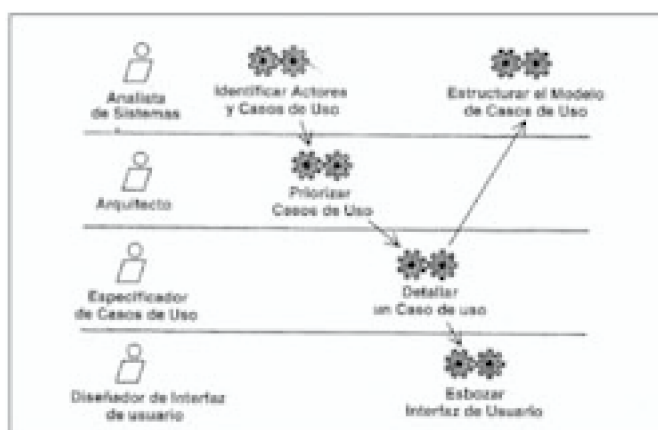


Figura 2□8: Un flujo de trabajo: entidades ejecutoras - actividades

Se pueden identificar las actividades asignadas a cada trabajador, las dependencias entre ellas y si algún trabajador participa en varias actividades simultáneamente. Se trata pues, de un tema fundamental en el desarrollo de software: la carga de trabajo. Un sistema de ordenamiento y planificación puede ayudar a la toma de decisiones gerenciales sobre todo en las primeras etapas, en donde se calcula la carga a asignarse a cada miembro del equipo de desarrollo: también podría servir para identificar errores (los llamados cuellos de botella) y ubicar posibles excesos de trabajo en algunos recursos y falta de balance en la línea.

Dos flujos de trabajo donde este panorama se torna crítico son el modelado de negocio y la gestión de los requisitos: ambos ocupan gran parte de las dos primeras fases de RUP. Una incorrecta planificación puede conducir a gestionar erróneamente todo el proceso en sí.

⁵ : Un caso de uso representa una funcionalidad del sistema, expresada como una interacción entre él y uno o varios actores [BRI99].

2.7.1. Modelado de negocios (business modeling)

Se debe tener en cuenta que al comenzar un proyecto de desarrollo de software se está partiendo desde cero, por más que el equipo desarrollador tenga mucha experiencia en el área del problema que pretende resolver con el producto a implementar.

Modelado de negocios [EP00] es la técnica con la que se pueden plasmar los procesos de negocio del cliente (del área específica de la organización cliente donde se va a implantar el producto de software) a través de los modelos que presenta UML, como herramienta de RUP. Permite expresar las actividades de negocio y su comportamiento en relación con los actores, con quienes interactúan dentro de los procesos.

Esto debe verse reflejado (verificado) en los casos de uso planteados, obteniéndose conocimientos sobre lo que hacen los usuarios antes, durante y después de la ejecución de su participación en el caso de uso.

El modelamiento visual permite apreciar las relaciones (que podrían denominarse incluso dependencias) establecidas entre los objetivos y alcances de los sistemas a desarrollar, con los procesos de negocio que pretenden automatizar. En el siguiente diagrama conceptual aclara estos conceptos:

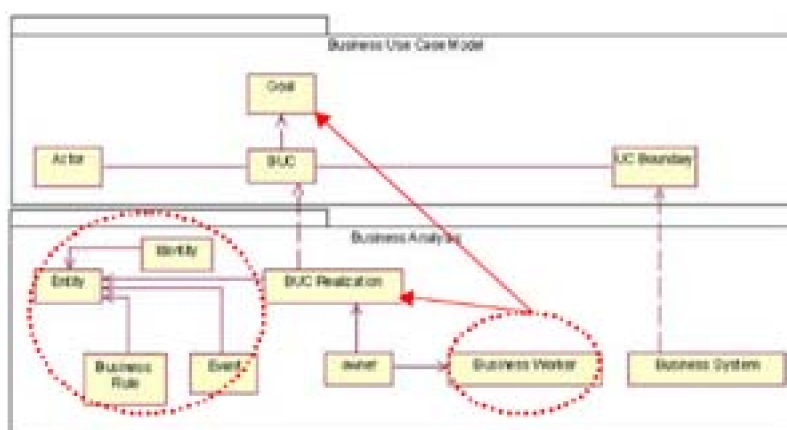


Figura 2□9: Conceptualización del modelado de negocios [Joh04]

En la figura se puede observar como un trabajador (*business worker*) realiza una operación de negocios pero que ésta depende tanto de otras entidades, como de eventos y reglas de negocios; todas ellas se organizan para la consecución de un objetivo. Claramente se enmarca la necesidad de tener una secuencia adecuada de las operaciones que están inmersas en los casos de uso, lo cual afectará la etapa que sigue: la captura de requisitos.

Siguiendo la metodología RUP, se pueden detectar dos ideas rectoras que afectan al desarrollo: costo y calidad. El modelado de negocios, como abstracción mínima del ambiente en donde se pretende aplicar la solución debe mostrar las actividades que permitirán cumplir los objetivos planteados en, por ejemplo, el documento de visión y alcance del sistema [Ale98]

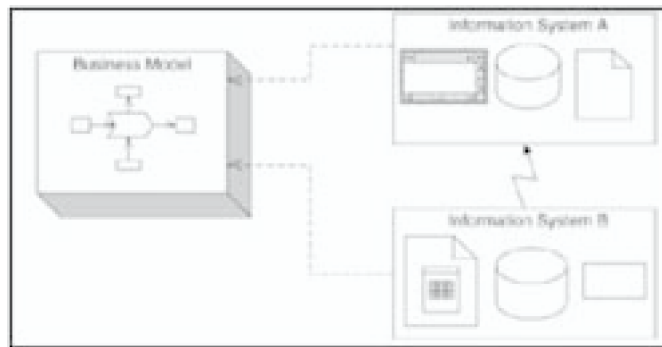


Figura 2□10: El modelado de negocios como base para la definición de requisitos de sistemas

Para lograr ese objetivo se hacen recomendaciones sobre la calidad del proceso de programación e incluso sobre la gestación inicial del proyecto, lo que involucra la identificación de elementos que se mencionaron en apartados anteriores tales como riesgos, objetivos dentro de la visión, el tema del esfuerzo, etc. Estas recomendaciones son hechas igualmente, en varias metodologías [Hig03] pero ninguna ofrece una forma de planificar las actividades de Ingeniería de Software inmersas dentro de este tipo de proyectos. Se puede ver esta situación en la siguiente figura:

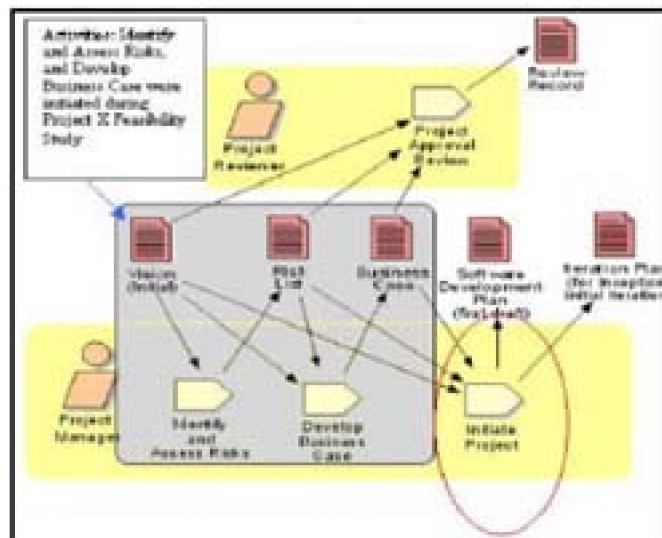


Figura 2□11: Tareas involucradas en el modelamiento de negocios

2.7.2. Captura de requisitos

La captura de requisitos es el paso fundamental en el levantamiento de la información necesaria para el desarrollo de un proyecto de Ingeniería de Software. El entendimiento de la naturaleza propia de cada problema que se pretende resolver por medio de un sistema de información es algo muy complejo: por lo mismo, el establecimiento de lo que el software debe hacer también lo es [Pre02].

Sommerville define a un requerimiento como un servicio que presta o una restricción

que cumple el sistema [So92], pero de inmediato indica que el término no es usado de forma adecuada en la industria del Software. Algunos autores explican que la confusión se debe a que lo que se necesita para solucionar un problema de esta naturaleza incluye la información del cliente – usuario (que se pueden denominar los requisitos del usuario), los requerimientos del sistema como tal y las especificaciones (alcances técnicos) propias del diseño de software [Dav93].

Los tres tipos de requerimientos están en niveles diferentes por lo que se ha de precisar también de distintas formas de captura. Se tienen entonces:

- Requerimientos del usuario en un nivel abstracto o de alto nivel
- Requerimientos del sistema: descripción detallada de lo que el sistema debe hacer.
- Especificación del diseño de software: base de diseño e implementación.

Determinar qué tareas es algo que las metodologías no mencionan con claridad, a pesar de la existencia de formatos para entrevistas, encuestas y demás ⁶.

2.7.2.1. Identificando los requisitos

Pressman indica cinco áreas de esfuerzo en donde se debe planificar con sumo cuidado las acciones involucradas en el levantamiento de información: reconocimiento del problema, evaluación y síntesis ⁷, modelado de los requisitos, especificación y revisión [Pre02]. Durante la comunicación con el cliente -a manera de entrevista- es que se van obteniendo los datos necesarios (requerimientos de usuario) y se recuerda que las preguntas son de contexto libre [Gau89]: reiteradamente se encuentra otro vacío que no permite planificar las entrevistas ni todo el proceso del análisis en sí, de los datos adquiridos. Aunque no se debe perder la perspectiva que un único planificador es imposible de diseñar, ante la multitud de variables involucradas en un proyecto de software.

Existen técnicas tales como la TFEA [Zah90] (técnicas para facilitar las especificaciones de la aplicación) que propugna la creación de un equipo de trabajo para la captura, negociación y administración de los datos necesarios para resolver el problema pero con la falta de un método que planifique u ordene el proceder de dichos grupos.

2.7.2.2. Los requisitos en RUP

RUP da por su parte, una definición para cualquier tipo de requisito ⁸ como sigue: un requerimiento describe una condición que el sistema debe cumplir o una capacidad que éste debe tener; debe estar derivado directamente de una necesidad expresa del usuario final y debe estar especificada, estandarizada o formalmente impuesta en algún tipo de

⁶ : RUP y PMI tienen sendos documentos para las entrevistas pero consideramos que se limitan a ser plantillas genéricas de títulos. En lo referido a las preguntas a hacerse, hay un gran vacío (N. del A.)

⁷ : Estas podrían darse en la etapa de definición del proyecto y podrían verse relegadas en el documento de visión (N. del A.)

⁸ : Se refiere a la diferencia entre requisitos funcionales y no-funcionales (N. del A.)

documento [BRI00] [Eel04] [Can03c].

Deben guiar el desarrollo hacia la construcción del sistema correcto por medio de una descripción suficientemente buena de los requisitos del sistema – que se vieron en el apartado anterior- que permita llegar a un acuerdo entre el cliente-usuario y los desarrolladores.

Parte de la receta presentada por Booch, Jacobson y Rumbaugh para su determinación incluye la enumeración de los requisitos candidatos, la compresión del contexto del sistema y la captura en sí. Se explica además que estos pasos no se dan separadamente, lo que indica que hay un nivel alto de dependencia y que el trabajo probablemente es realizado por varios actores (se esta haciendo referencia a los miembros del equipo de desarrollo)

Se presenta la necesidad de ordenar estas tareas por prioridades de captura, por prioridades de ejecución, por niveles de riesgo, etc.; a su vez, los ejecutores de estas actividades tienen diferentes rendimientos por lo que encaja perfectamente en el planteamiento del problema del scheduling, tal y como se lo presentó en apartados anteriores.

2.8. Algunas metodologías de planificación de proyectos complementarias a RUP: estado del arte de la planificación de proyectos de Software

2.8.1. Planificación en proyectos usando COST: comercial off-the-shelf

El desarrollo de proyectos de sistemas de información que incluyan algún tipo de componente (libre o licenciado) como herramientas de desarrollo, librerías, compiladores, APIs, etc. está constituyendo una estrategia moderna muy empleada [MS00]; una de ellas es denominada COST. En vez de implementar la totalidad de componentes del sistema, la estrategia COST promueve el uso de componentes previamente diseñados, y aplicando RUP, poder reducir así el tiempo de determinadas fases del ciclo de desarrollo.

Con los riesgos que puede implicar junto con el ahorro de tiempo, su aplicación requiere de un nivel de ordenamiento dado para ciertas tareas que ahora presentarán dependencia entre sí y estarán subordinadas al correcto funcionamiento de los componentes COST. Chan [Cha03] afirma desde la perspectiva de RUP, que deben verse estos componentes pre fabricados como fases terminadas (o avanzadas) ya desde las primeras planificaciones (fase de concepción).

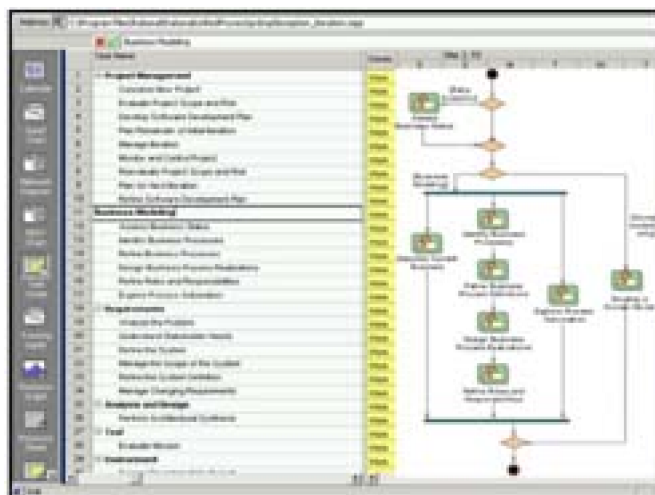


Figura 2-12: Incluyendo los componentes COST como parte del desarrollo [Cha03]

2.8.2. UED: Unifying Enterprise Development Teams

Booch [Boo00] presenta aquí una formulación particular para equipos de trabajo inter-empresa (áreas de sistemas propias y no consultoría externa) para el desarrollo de proyectos Web, manifestando la necesidad de una secuencialización adecuada de los procesos de negocios que se ven reflejados en los casos de uso respectivos. A este proceso de ordenamiento se ha hecho referencia en apartados anteriores en donde se puede observar la importancia no sólo de un correcto planteamiento de los casos de uso –solicitados por RUP, sino también de su correcto ordenamiento en el tiempo dentro del entorno organizacional.

2.8.3. IBM-Rational Unified Process: IBM – RUP

West [We02] plantea el punto de vista de IBM para el desarrollo y la aplicación de RUP en proyectos de software y señala como aspectos fundamentales de la administración de cualquiera de ellos a lo siguiente:

- Las actividades definidas deben ayudar al administrador a construir un plan ordenado, respetando dependencias, presupuestos y tiempos.
- El factor de comunicación entre fase cuyas salidas se comportan como entradas de las fases siguientes es fundamental (otra forma de resaltar dependencias entre fases)
- Los actores que ejecutan las actividades programadas deben estar organizados de tal forma que se eviten sobrecargas y cruces (saber siempre quién hace qué).

2.8.4. eXtreme Programming: RUP para pequeños proyectos

eXtreme Programming (XP) es una disciplina de desarrollo de software desarrollada por Kent Beck en 1996 la cual consta de cuatro pilares: comunicación, simplicidad, feedback,

y determinación en la toma de decisiones [Pol01]. Está centrado en la programación (code-centric) y como RUP, presenta varias etapas:

- Pequeños prototipos basados en una lista de tareas y de funcionalidades que deben ser cubiertas.
- Diseño simple
- Pruebas y validación de los prototipos
- Integración constante

XP requiere de planes de proyecto algo menos detallados que en RUP debido a que es habitual su uso en escalas pequeñas pero igualmente es necesario un ordenamiento de las tareas para buscar la minimización del tiempo de desarrollo de prototipos sin funcionalidades que han de ser definitivas [Bec00].

2.8.5. MSP: Integrando Managing Successful Programmes con RUP

Managing Successful Programmes (siglas MSP) provee un conjunto de buenas prácticas de carácter no-propietario (es decir, que no es una metodología perteneciente a una empresa, por lo que no estaría amarrada a ningún producto en general ni de análisis – framework, ni de diseño o implementación) desarrollado por la Oficina de Control Gubernamental del Gobierno Británico [Nor05] para proyectos de implementación de software tanto públicos como privados (o third-part).

Combinando RUP y esta metodología se pueden obtener mejores resultados en la administración del proyecto y en el control de la configuración del cambio. La siguiente figura muestra una forma de complemento entre RUP y MSP:



Figura 2-13: Complementándose: RUP y MSP [Nor05]

Algunos de los elementos de MSP son los siguientes:

- Determinación de un alcance formal sobre la estructura del negocio –cliente- y los procesos que se ven y verán afectados por la construcción del software, a manera de la visión del desarrollo.
- Métodos de captura y **ordenamiento de los requisitos de negocio.**
- Métodos para adaptar las fases del desarrollo a la medida del cliente.
- Determinación de las mejores herramientas para identificar, capturar y analizar riesgos y esfuerzos requeridos.
- Presencia de plantillas de documentación.

Como se puede apreciar, se guarda gran similitud con algunos de los planteamientos que tiene RUP por lo que se ha resaltado la importancia de un método de ordenamiento de los requisitos (visualizados como casos de uso de negocio – futuras funcionalidades del sistema) que pueda manejar los temas tanto de dependencias como de los actores involucrados en su ejecución.

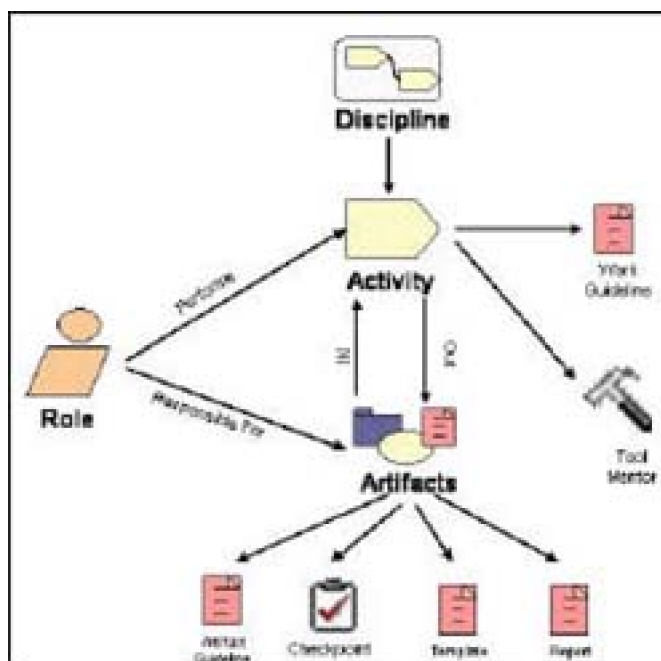


Figura 2-14: Disciplinas RUP – MSP y los roles de las entidades ejecutoras

MSP se presenta especialmente como una metodología de planificación de la implementación de cambios de negocios a partir de la construcción (entrada en la organización cliente) de un sistema de información. El ciclo de vida planteado por MSP también requiere de algún tipo de planificador:

- Fase de inicialización (comparable con concepción de RUP): sus procesos involucrados serían:
 - Establecimiento y procesamiento de objetivos
 - Identificación de los requisitos de software y de negocio: claro entendimiento de las políticas organizacionales.⁹

- Definición de la arquitectura del sistema.
 - Planificación de las tareas en el entorno.
- Fase de ejecución: la que incluye
- Administración del proyecto: manejo de tiempos, costos y calidad.
 - Administración de riesgos
 - Diseño e implementación: manejo de métricas de desarrollo.
 - Finalización del proyecto

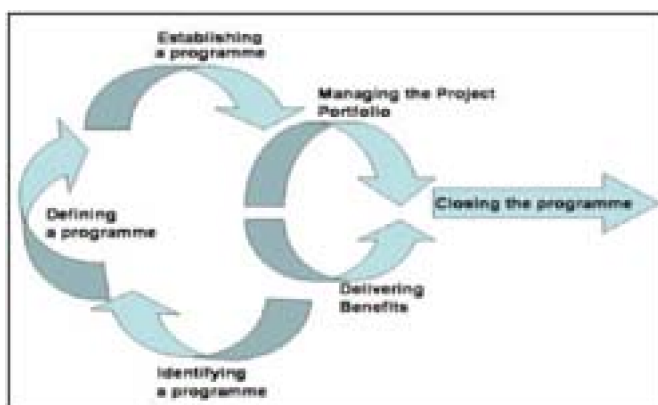


Figura 2-15: Ciclo de vida MSP

2.8.6. Estimación de proyectos con IBM Rational SUMMIT Ascendant

IBM Rational SUMMIT Ascendant [Kro04] es a una técnica y a la vez una suite de productos que consiste en herramientas para la planificación y estimación para proyectos Web (interfase). Planteado por IBM, ayuda a estimar los costos y jerarquizarlos así como los planes preliminares a establecerse en las primeras etapas RUP a manera de complemento.

La estimación y la planificación usando herramientas SUMMIT permiten planificar las tareas involucradas en proyectos Web, por iteraciones mostrando a los administradores de proyecto dos aspectos del desarrollo:

- Trabajos que necesitan ser hechos o mejorados (de acuerdo a la fase RUP en la que se está) y las múltiples subordinaciones entre las operaciones.
- La efectividad del equipo de trabajo: control de calidad y métricas de software.

Provee además diversas técnicas de estimación aplicables a los componentes desarrollados como son COCOMO II, puntos de función [Pow04], QIF, etc. SUMMIT Ascendant divide la planificación del proyecto en tres actividades principales:

⁹ : A esto lo denomina la metodología *programme*: acepción británica para requisito de proyecto (N. del A.)

- Medición inicial del tamaño del proyecto: determinando las actividades a realizarse (actividades de star-up) para luego ordenarlas por prioridades (objetivos – funcionalidades – dependencias)
- Planificación de la fase (de RUP) completa, indicando las iteraciones: la planificación también incluye estimaciones de costo y esfuerzo.
- Planificación de cada iteración como una guía progresiva de trabajo y asignación: programación de ejecutores de las actividades involucradas en la planificación.

En la siguiente figura se plantea un ejemplo de un plan trazado con herramientas SUMMIT:



Figura 2-16: Herramienta para esgrimir planes en IBM-Rational SUMMIT

Mientras que un ejemplo de estimación se aprecia a continuación:



Figura 2-17: Estimación de costo – tiempo- esfuerzo en SUMMIT

No consta en la literatura el uso de algoritmos heurísticos y meta heurísticos para la realización del ordenamiento de las tareas dependientes en esta suite de herramientas.

2.8.7. Project Portfolio Management (PPM): alineando negocios con Tecnologías de Información -TI

El incremento del uso de TI en casi cualquier tipo de negocio – sea producción o dotación de servicios- hace necesaria la planificación de la intervención de esta tecnología en los proyectos que se le requieran; precisamente, sin una adecuada planificación y gestión de la tecnología es que se producen grandes fallas en los desarrollos, generando temor en el uso de las emergentes TI. Una estrategia soportada por IBM Software Development es la denominada Project Portfolio Management (PPM) [Red04] la cual ayuda a las

empresas a enfrentar estos problemas de administración, especialmente en proyectos de software que han sido desarrollados con RUP. PPM es una estrategia que permite a los administradores de proyectos alinear sus políticas corporativas a las TI que formarán parte del plan de desarrollo. En esencia PPM permite, por medio de herramientas visuales, el manejo de un portafolio de proyectos (o un portafolio de fases del proyecto: juntando desarrollo de software con administración de negocio) analizando riesgos individuales de las fases y los objetivos.

Algunas de las responsabilidades de los actores ejecutores de un proyecto que aplica PPM se pueden ver en la siguiente lista:

Role	Challenges
EP/PO/CS/CTO	• Cope with reduced budgets and increased expectations. • Meet productivity goals consistently. • Align business goals and IT projects. • Use reliable measures to determine whether teams are really working on the "right" projects. • Put out fires and cut costs that prevent positive planning.
Portfolio Program Manager	• Prioritize initiatives, resources, and assets across the project portfolio. • Assess and communicate portfolio status. • Ensure consistent processes across projects. • Reallocate resources and optimize key resources across projects.
Project Manager	• Align project management and application development processes. • Match up business teams and technical team perspectives. • Revisit project outcomes, assess project status, and identify inter-project dependencies. • Manage scope, planning, verification, and change management.
Team Members	• Understand project scope, priorities, requirements, dependencies, and traceability. • Monitor and report progress without sacrificing coding activities.

Figura 2-18: Roles y responsabilidades de los actores del proyecto

Para la aplicación del framework que brinda PPM es necesario un ordenamiento de los requisitos obtenidos en el modelado de negocio RUP por lo que también se puede notar la necesidad de una estrategia de ordenamiento y asignación de trabajos.

2.9 Conceptos generales sobre heurísticas y meta heurísticas

Las técnicas heurísticas y meta heurísticas constituyen métodos aproximados que ofrecen soluciones suficientemente buenas a este tipo de problemas; es decir, no exactas pero aproximadas a dichas soluciones como para resolver el problema con costos computacionales relativamente bajos. Las heurísticas son métodos algorítmicos exploratorios aplicados a la resolución de problemas en los cuales las soluciones se descubren por la evaluación progresiva de un conjunto de soluciones o candidatas hacia un resultado final sin analizar todas las posibles combinaciones [IEEE03].

Los procedimientos meta heurísticos son una clase de métodos aproximados que están diseñados para resolver problemas difíciles de optimización combinatoria, en los que las soluciones heurísticas clásicas no son efectivos. Los métodos meta heurísticos proporcionan un marco general para crear nuevos algoritmos híbridos combinando diferentes conceptos derivados de la Inteligencia Artificial (en adelante IA), la evolución biológica y los mecanismos estadísticos.

La técnica meta heurística utilizada en el presente trabajo de tesis, para implementar un algoritmo que planifique las tareas de un proyecto de desarrollo de software es la

técnica GRASP (del inglés Greedy Randomized Adaptative Search Procedure). Esta técnica fue desarrollada por T. Feo y M. Resende a finales de los años 80 [FR95].

Mientras que el criterio voraz permitía seleccionar solamente el mejor valor de la función objetivo a optimizar, los algoritmos GRASP relajan o amplían este criterio de tal manera que, en vez de seleccionar un único elemento, forma un conjunto de elementos candidatos a ser parte del conjunto solución y que cumplen ciertas condiciones. Es sobre este conjunto formado, que realizará una selección aleatoria de algún elemento.

Tienen esta denominación por lo siguiente:

- Son procedimientos de búsqueda: dentro de un espacio de posibles soluciones se realizan búsquedas sin evaluar a todos los elementos del problema.
- Son voraces: porque en cada evaluación escoge a los mejores candidatos que cumplan ciertas condiciones.
- Son de progresiva adaptación: porque se adapta a la estructura de la instancia del problema que pretende resolver.
- Son aleatorias: porque de entre las candidatas escoge aproximadamente al azar, aquellas que finalmente formarán parte de la solución

Esta ampliación del número de candidatas sumadas a la aleatoriedad de la selección, permite tener un mayor espectro de soluciones posibles logrando acercarse más a una probable solución óptima del problema, caso exista.

La siguiente figura presenta el algoritmo GRASP general:

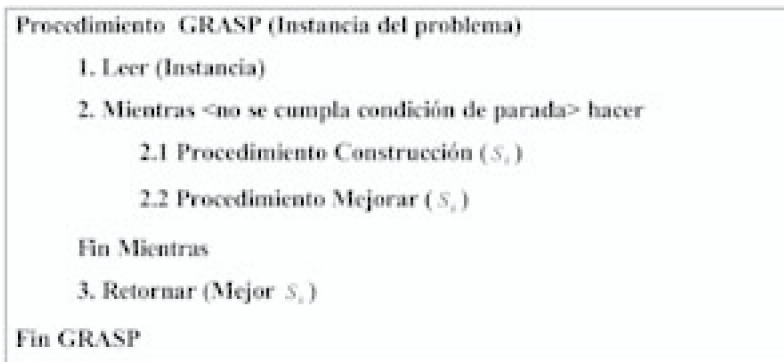


Figura 2-19: Estructura general del algoritmo GRASP

Sobre este algoritmo podemos afirmar:

- Línea 1: se ingresan los datos que conforman la instancia del problema
- Línea 2: el proceso GRASP seguirá mientras no se cumpla una condición de parada. La condición de parada puede ser de varios tipos: optimalidad (el resultado obtenido presenta cierto grado de aproximación a la solución exacta o es lo suficientemente óptimo); número de ejecuciones realizadas (cantidad de iteraciones); tiempo de procesamiento (el algoritmo será ejecutado durante un determinado período de tiempo)

Líneas 2.1 y 2.2: se ejecutan las dos fases principales de un algoritmo GRASP, a seguir:

etapa de construcción de la solución golosa adaptada y aleatoria; y la etapa de mejoría de la solución construida anteriormente.

Línea 3: finalmente se devuelve el resultado de la aplicación del algoritmo.

Debemos tener en cuenta que al finalizar la presente fase de construcción ya se tiene una instancia solución al problema, aunque su condición o grado de optimalidad puede distar del óptimo. Probablemente una ejecución aleatoria no arroje un mejor valor que el caso de la aplicación del algoritmo voraz; sin embargo, se ha demostrado que numerosas ejecuciones de esta fase pueden generar una solución mejor.

Como en los casos de las heurísticas, las soluciones construidas por medio de algoritmos GRASP no son necesariamente óptimas locales, por lo que se hace necesaria la aplicación de un procedimiento de mejora de la solución encontrada en la fase de construcción. Es considerada como una fase de post procesamiento y su aplicabilidad depende de las restricciones del problema.

2.9.1. Fase de construcción GRASP

$$\begin{aligned}\beta &= \text{Mejor}\{c(x) : x \in N\} \\ \tau &= \text{Peor}\{c(x) : x \in N\}\end{aligned}$$

Figura 2-20: Valores extremos de la función

El conjunto de candidatas RCL (del inglés: Restricted Candidates List): se formará con todos los elementos x de N que cumplan con la condición:

$$RCL = \{x \in N : \beta \leq c(x) \leq \beta + \alpha(\tau - \beta)\}$$

Al parámetro α que aparece en la definición de RCL se le conoce como parámetro de relajación y es el responsable de tornar la solución que surgirá, más amplia que la solución voraz.

Del conjunto RCL se toma un elemento al azar que pasa a formar parte del conjunto S , repitiéndose el proceso hasta que se hayan procesado todos los elementos de N . Una estrategia aleatoria viene a ser particularmente útil cuando existen varias formas para realizar un paso dentro de un algoritmo en general, y donde resulta difícil garantizar totalmente que alguna de ellas es la mejor elección. Eso ocurre con el criterio goloso al no analizar más allá los efectos de la selección del mejor elemento de c en cada iteración.

El criterio GRASP indica que el candidato a ser parte del conjunto solución es así, un elemento seleccionado en forma aleatoria desde el conjunto RCL (no pertenece a S y su valor de la función objetivo no necesariamente es el mejor en ese instante). Como se puede apreciar, depende del valor de α empleado. Según el valor que tome α , el

comportamiento del criterio GRASP puede variar:

$\alpha = 0$	→	Criterio totalmente aleatorio
$\alpha = 1$	→	Criterio totalmente goloso
$0 < \alpha < 1$	→	Criterio GRASP convencional que requerirá de calibración adecuada a cada problema.

Figura 2-21: Posibles valores de la constante de relajación α

La siguiente figura muestra al algoritmo detallado de la fase de construcción:

```
Procedimiento Construcción( $N, c, E, F, S, \hat{\alpha}$ )
Inicio
  Leer  $c, E, \hat{\alpha}$ 
   $S = \emptyset$ 
   $N = E$ 
  1. Mientras <no se cumpla condición de parada> hacer
    Inicio
      1.1  $RCL = \emptyset$ 
      1.2  $\beta = \text{Mejor}\{c(x) : x \in N\}$ 
      1.3  $\tau = \text{Peor}\{c(x) : x \in N\}$ 
      1.4  $RCL = \{x \in N : \beta \leq c(x) \leq \beta + \alpha(\tau - \beta)\}$ 
      1.5  $u = \text{Aleatorio}(RCL)$ 
      1.6 Si  $S \cup \{u\} \in F \Rightarrow S = S \cup \{u\}$ 
      1.7  $N = N - \{u\}$ 
      1.8 Adaptar  $c$ 
    Fin Mientras
  Fin Construcción.
```

Figura 2-22: Fase de construcción GRASP

Algunos comentarios:

- Línea 1: el proceso se repetirá hasta completar una condición de parada adecuada.
- Líneas 1.1-1.3: se inicializa en vacío la lista RCL y se hallan el mejor y peor valor de c en N .
- Línea 1.4: se forma la lista RCL siguiendo el criterio GRASP relajado
- Línea 1.5: se escoge de forma aleatoria un elemento de RCL.
- Línea 1.6: se determina si es viable el conjunto solución agregándole el elemento seleccionado en la línea anterior.
- Línea 1.7: se elimina el elemento seleccionado de N
- Línea 1.8: se adapta la función c para las siguientes iteraciones.

Téngase en cuenta que al finalizar la presente fase de construcción ya se tiene una instancia solución al problema, aunque su condición o grado de optimalidad puede distar del óptimo. Probablemente una ejecución aleatoria no arroje un mejor valor que el caso de la aplicación del algoritmo voraz; sin embargo, se ha demostrado que numerosas ejecuciones de esta fase pueden generar una solución mejor.

2.9.2. Fase de mejoría de la solución GRASP

Como en los casos de las heurísticas, las soluciones construidas por medio de algoritmos GRASP no son necesariamente óptimas locales, por lo que se hace necesaria la aplicación de un procedimiento de mejora de la solución encontrada en la fase de construcción. Es considerada como una fase de post procesamiento y su aplicabilidad depende de las restricciones del problema.

Siendo la GRASP una generadora de buenas soluciones, se espera que al menos una de ellas este dentro del entorno o vecindad (F) de la solución óptima [La91]. Lo primero que establece esta fase de mejoría entonces, es la definición de una vecindad de soluciones alrededor de la solución hallada en la fase de construcción. Esta vecindad puede ser determinada intercambiando uno a uno los elementos de la solución con aquellos que no pertenecen a la solución, siempre que se verifique la propiedad F.

La forma más sencilla de trabajo es limitar la búsqueda a k-vecindades. Dado un vector binario $X = \{X_1, X_2, X_3, \dots, X_N\}$ y un entero positivo k, definimos el conjunto:

$$N_k(X) = \{z_1, z_2, z_3, \dots, z_k \mid \sum_{j=1}^n |z_j - x_j| \leq k, z_j \in \{0,1\}\}$$

Es denominado k-vecindad de X e incluye los elementos que son distintos de X en no más k componentes. Un intercambio con k permutaciones, permitiría mover un punto i-ésimo de X hacia otro punto i+1-ésimo hasta que no se produzcan mejoras en la calidad de las soluciones que aporta X. La siguiente figura muestra la fase de mejora GRASP:

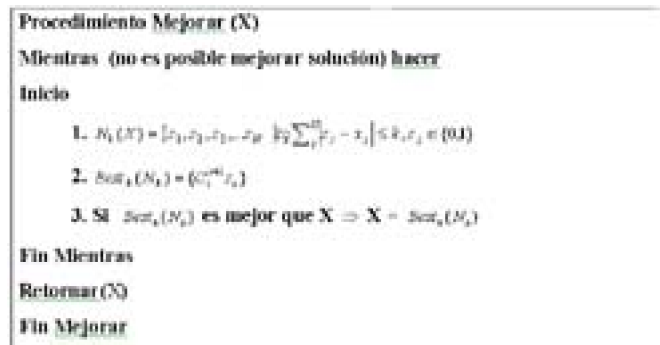


Figura 2-23 : Fase de mejoría GRASP

Debemos notar en este algoritmo lo siguiente:

- Línea 1: se forma la vecindad k-ésima alrededor de la solución X en donde cada elemento de N_k tiene una diferencia (grado de separación) máxima igual a k.
- Línea 2: se halla el mejor elemento de la vecindad a través de la aplicación de combinaciones de los elementos.
- Línea 3: si se logra mejorar X, los siguientes refinamientos serán sobre esta solución mejor.

Capítulo III: Un modelo matemático de optimización para el problema del task scheduling en la planificación de los recursos en los proyectos de Ingeniería de Software

3.1. Elementos del modelo. Consideraciones generales

En este capítulo se muestra un modelo de optimización para la solución del problema de la planificación de recursos en un proyecto de Ingeniería de Software. Considérese los siguientes enunciados:

- Sea N el número de tareas o actividades dependientes involucradas dentro del proyecto de software rotuladas como $J_1, J_2, J_3, \dots$

- Sea M el número de recursos (EP2) (trabajadores) heterogéneos disponibles, rotulados como $M_1, M_2, \dots, M_M$. Los recursos ejecutarán las N actividades del proyecto.
- Sea T_{ik} el tiempo que demora el recurso k -ésimo en ejecutar la tarea i -ésima donde

$$k \in 1 \dots M, i \in 1 \dots N$$

Esta información se dispondrá en un formato matricial. Es posible que una tarea no pueda ser ejecutada por algún recurso debido a algún tipo de restricción particular del rol del recurso; en tal caso se considerará un tiempo de ejecución muy alto: $T_{ik} = \infty$

- Sea p_{ik} el indicador si la tarea J_k es predecesora de la tarea J_i y en donde se cumple que:

$$P_{ik} = \begin{cases} 1 & \text{si la tarea } J_k \text{ es predecesora de la tarea } J_i \\ 0 & \text{en caso contrario} \end{cases}$$

Sea U_k el tiempo de finalización de la ejecución de la tarea J_k

- Sea V_k el tiempo de inicio de la ejecución de la tarea J_k

3.2. Variables de decisión del modelo

Sea X_{ik} una tarea i -ésima ha ser ejecutada por un recurso k -ésimo en donde se cumple que:

$$X_{ik} = \begin{cases} 1 & \text{si la tarea } i\text{-ésima es ejecutada por el recurso } M_k \\ 0 & \text{caso contrario} \end{cases}$$

Sea A_k tiempo de trabajo (procesamiento) acumulado del recurso M_k

Sea el makespan =:

$$\max_{1 \leq j \leq M} \left\{ \sum_{i=1}^n T_{ij} * X_{ij} \right\}$$

la duración total del proceso de ejecución de las actividades (función objetivo). En el modelo, haremos referencia al makespan por medio de la variable X_0

3.3. Restricciones del modelo

El modelo va a presentar cuatro restricciones. Una tarea puede tener 0 o más

predecesoras. Cada tarea debe ser ejecutada una sola vez, por un solo recurso ya que no hay división de operaciones dentro de cada tarea. Además se deben atender las predecesoras de cada tarea antes de programarla. Finalmente, mientras cualquier recurso esté ocupado, no puede ejecutar en paralelo otra actividad hasta terminar con la actual.

La restricción de asignación

$$\sum_j^M X_{ij} = 1 \quad \forall i \in 1..N$$

indica que cada tarea es asignada una vez a un recurso durante todo el proceso de programación

Por su parte, la restricción de precedencia

$$\sum_{r=1}^M X_{ir} \leq \sum_{r=1}^M X_{kr} \quad \forall P_{ki} = 1 \quad k \in 1..M$$

expresa que la tarea J_i será programada obligadamente después de que todas sus predecesoras J_k hayan sido ejecutadas.

La restricción sobre el tiempo acumulado de trabajo

$$\sum_{i=1}^N T_{ij} - A_j = 0 \quad \forall j \in 1..M$$

muestra que la suma de los tiempos de ejecución de cada tarea i -ésima asignada al recurso j -ésimo, debe coincidir con su tiempo acumulado en el vector A .

De otro lado, la restricción sobre el makespan y

$$X_0 - A_j \geq 0 \quad \forall j \in 1..M$$

indica que éste valor es el mayor de los tiempos de trabajo acumulado de todos los recursos del proyecto.

3.4. El modelo de optimización matemática para la variante desarrollada

El modelo matemático para la variante desarrollada se puede apreciar en la siguiente figura:

Figura 3-1: Modelo matemático para la variante del task scheduling desarrollada

3.4.1. Comentarios sobre el modelo presentado: relajación

El modelo presentado tiene un carácter de relajado porque presenta algunas restricciones adicionales internas. Una restricción temporal adicional, en donde se cumple que para toda tarea J_i :

$$V_i = \min\{T_{ik} + \max\{A_k, \max\{U_r : \forall P_{ri} = 1\}\}\}$$

$$U_r = \min\{T_{ik} + \max\{A_k, \max\{U_r : \forall P_{ri} = 1\}\}\} \quad r \in [0, M], k \in [1, M], \forall i \in [1, N]$$

Así se indican dos aspectos de la asignación:

Que la tarea J_i será ejecutada a partir del tiempo de finalización de la última de sus predecesoras en ejecutarse.

Que para cada J_i tarea, obtenemos los M valores de:

$$T_{ik} + \max\{A_k, \max\{U_r : \forall P_{ri} = 1\}\}$$

y determinamos el mínimo, el cual será el tiempo de finalización de la ejecución de dicha tarea. La expresión

$$T_{ik} + \max\{A_k, \max\{U_r : \forall P_{ri} = 1\}\}$$

representa la suma de la entrada de la matriz de tiempos de ejecución T_{ik} con el máximo de dos valores:

O el tiempo acumulado del recurso k -ésimo, A_k

O el mayor tiempo de finalización de todas las predecesoras de J_i

El modelo no contempla directamente estas últimas restricciones relativas al tiempo de procesamiento de las tareas después de todas sus predecesoras. A pesar de esta situación constituirá una buena medida para probar la eficiencia real de los algoritmos propuestos y qué tan cerca están de la solución exacta.

3.5. Importancia del modelo matemático

El modelo matemático ha sido planteado para determinar la calidad y eficiencia real de los algoritmos propuestos en la presente tesis. En la realidad se presentan factores que afectan la dependencia de las actividades implicadas en los proyectos de software, pudiendo flexibilizarla. Igualmente, muchos de los aplicativos existentes (MS Project, Primavera Project, etc.) contemplan el tema de la sobrecarga de recursos –humanos- al ejecutar las tareas con el correspondiente riesgo que esto pueda producir. En un modelo matemático más estricto, este tipo de situaciones no pueden ser contempladas por lo que se consideraría que no es tan apegado a la realidad de las líneas de desarrollo de software.

Con la presencia del modelo se busca conseguir los resultados exactos para instancias del problema y contrastarlos con las soluciones arrojadas por los algoritmos heurísticos voraz y GRASP a presentarse en el siguiente capítulo.

El modelo va a ser pasado a aplicaciones que resuelven problemas de programación no lineal como lo es el programa LINDO. Vamos a presentar un ejemplo para una instancia de prueba pequeña¹⁰: sean $N = 6$ tareas y $M = 3$ recursos, se tiene la siguiente matriz de tiempos T :

	M1	M2	M3
J1	10	12	10
J2	11	13	20
J3	25	24	23
J4	10	11	10
J5	9	10	20
J6	23	9	8

Tabla 3-1: Una matriz de tiempos T para una instancia 6x3

Luego también se tiene la lista de predecesoras por tarea:

	Predecesoras
J1	No tiene
J2	J1,J4
J3	J5,J6
J4	J6
J5	J6
J6	No tiene

Tabla 3-2: Lista de predecesoras para una instancia 6x3

¹⁰ : Las justificaciones sobre la calidad de los algoritmos se verán en el capítulo correspondiente a los experimentos numéricos.

El modelo ejecutado en el programa LINDO sería:

```
MIN X0
! RESTRICCION DEL MAKESPAN, ES EL EJEMPLO PRESENTADO EN
EL TEXTO DE LA TESIS
SUBJECT TO
    10 X11 + 11 X21 + 25 X31 + 10 X41 + 9 X51 + 23 X61 - A1 = 0
    12 X12 + 13 X22 + 24 X32 + 11 X42 + 10 X52 + 9 X62 - A2 = 0
    10 X13 + 20 X23 + 23 X33 + 10 X43 + 20 X53 + 8 X63 - A3 = 0
    X0 - A1 >= 0
    X0 - A2 >= 0
    X0 - A3 >= 0
! RESTRICCION DE ASIGNACION
    X11 + X12 + X13 = 1
    X21 + X22 + X23 = 1
    X31 + X32 + X33 = 1
    X41 + X42 + X43 = 1
    X51 + X52 + X53 = 1
    X61 + X62 + X63 = 1
! RESTRICCION DE PREDECESORAS
    X21 + X22 + X23 - (X11 + X12 + X13) <= 0
    X31 + X32 + X33 - (X41 + X42 + X43) <= 0
    X31 + X32 + X33 - (X51 + X52 + X53) <= 0
    X31 + X32 + X33 - (X61 + X62 + X63) <= 0
    X41 + X42 + X43 - (X61 + X62 + X63) <= 0
    X51 + X52 + X53 - (X61 + X62 + X63) <= 0
END
INT X11
INT X12
INT X13
INT X21
INT X22
INT X23
INT X31
INT X32
INT X33
INT X41
INT X42
INT X43
INT X51
INT X52
INT X53
INT X61
INT X62
INT X63
```

Figura 3-2: Modelo en LINDO para $N=6$, $M=3$

Capítulo IV: Algoritmos Propuestos

4.1. Alcances y limitaciones de la solución a plantearse

Se ha desarrollado un algoritmo voraz y un algoritmo GRASP para resolver la variante especificada del problema sobre la base de un doble criterio: criterio de selección de la mejor tarea disponible para su ejecución y criterio de selección del recuso más idóneo y disponible para su ejecución. Para esto se está partiendo de las definiciones básicas que exige el modelo como son:

- Entradas del sistema: tareas, lista de predecesoras por tarea, recursos, tiempos de ejecución, información intermedia, etc.
- Salidas: makespan, lista de tareas que ha sido asignada a cada recurso, tiempo de trabajo y tiempo ocioso de cada recurso, etc.

Entre las limitaciones se indican:

- Las tareas que no tienen predecesoras se ejecutan en primer orden.
- Se esta considerando que el conjunto de tareas son para la creación o elaboración de

un único producto.

- Un recurso solamente puede procesar una tarea por vez.
- Los recursos no van a sufrir modificaciones de rendimiento durante todo el proceso de programación, es decir no va a modificarse los tiempos de ejecución por tarea.
- No se ha implementado la fase de mejoría de la solución GRASP debido a que en las investigaciones preliminares no produjo mejora alguna sobre los resultados de la fase de construcción precedente. Además, para efecto del análisis comparativo de la eficiencia, esta etapa no es requerida.

4.2. Estructuras de datos empleadas

Debemos partir del supuesto que se tiene una instancia de trabajo completa para la programación, que incluya estos datos:

- Cantidad de tareas y recursos: N y M respectivamente.
- Matriz de tiempos de ejecución T
- Lista de predecesoras por tarea.

Desde allí se esbozan las siguientes estructuras de datos:

N:	número de tareas $J_1, J_2, \dots, J_N$
M:	número de recursos $M_1, M_2, \dots, M_M$
Matriz T:	$[T_{ij}]_{N \times M}$ de tiempos de procesamiento, donde cada entrada representa el tiempo que se demora el recurso j-ésimo en ejecutar la tarea i-ésima.
Vector A:	$[A_i]$ de tiempos de procesamiento acumulados, donde cada entrada A_i es el tiempo de trabajo acumulado del recurso M_i
P_i :	Conjunto de tareas predecesoras de la tarea J_i
Vector U:	$[U_i]$ de tiempos de finalización de cada tarea J_i
Vector V:	$[V_i]$ de tiempos de finalización de las tareas predecesoras de J_i , donde se cumple que $V_i = \max\{U_j J_j \in P_i\}$
S_i :	Conjunto de tareas asignadas al recurso M_i
E:	Conjunto de tareas programadas
C:	Conjunto de tareas candidatas a ser programadas

Figura 4-1: Estructuras de datos necesarias

El llenado de estructuras tales como la matriz de tiempos T y las inicializaciones de los vectores A, S, U y V respectivamente, se pueden hacer de varias formas dependiendo de la implementación que se haga. Presentamos a continuación una forma general que se incluye en el algoritmo y que constituye la definición de las estructuras:



Figura 4-2: Proceso de definición y llenado de las estructuras de datos

Como se puede apreciar en la figura, es un proceso simple de llenado de las diferentes estructuras con valores arbitrarios. Este proceso será utilizado (invocado) directamente en el algoritmo, más adelante.

4.3. Lineamientos generales del algoritmo voraz propuesto

Los lineamientos generales del algoritmo recaen en tres actividades fundamentales: elegir la tarea que corresponda ser programada en un momento determinado; asignar su programación a una máquina disponible y la actualización de las estructuras de datos caso sufran modificación. Se sigue básicamente estos principios:

- Seleccionar las tareas aptas o candidatas con:
 - Un criterio voraz de selección de la mejor tarea a ejecutar: se escoge aquella tarea cuyo tiempo de ejecución sea el más bajo para evitar perturbaciones.
 - Un criterio voraz de selección del mejor recurso ejecutor disponible: la tarea se asigna al recurso que la puede ejecutar en menor tiempo.
 - Una tarea se considerará apta para ser ejecutadas si todas sus predecesoras ya han sido programadas previamente o si carece de ellas.
- En el caso particular de las tareas cuyas predecesoras ya han sido ejecutadas en su totalidad seguiremos así:
 - Como la tarea J_i se debe ejecutar después de todas sus predecesoras, debemos ubicar la tarea predecesora que termina su ejecución en el mayor tiempo y actualizar temporalmente su correspondiente vector V. La formulación que muestra la búsqueda de ese valor será:

$$V_k = \max\{U_r\}, J_r \in P_k$$

- Para determinar la máquina más adecuada para ejecutar una tarea J_k en particular se procede de la siguiente forma: a cada entrada del vector A se le suma V_k , o sea el tiempo de finalización de la última predecesora de esa tarea; finalmente, se le suma también el tiempo de ejecución T_{ik} . Se ubica la menor entrada de A (A_i) y esa máquina i-ésima es escogida.
- A la esta mejor máquina le asignamos tarea J, actualizando al vector A en la entrada correspondiente a M_k .
- Se repite el proceso para el resto de tareas
- Finalmente el mayor tiempo o entrada de la lista A será el makespan.

4.3.1. Ejemplo de aplicación: criterio voraz para la selección de la mejor tarea

Sean por ejemplo, las tareas J_1 y J_2 sin predecesoras; se tiene además, 3 recursos M_1 , M_2 , M_3 y un extracto de la matriz de tiempos de ejecución T en la tabla siguiente:

	M1	M2	M3
J1	10	12	11
J2	23	9	8

Tabla 4-1: Extracto de la matriz de tiempos de ejecución T

Sin perder generalidad y basándonos la tabla anterior se procede a formar dos listas ¹¹ con los tiempos de ejecución:

- Lista para la tarea J_1 : {10, 12, 11}. Determinamos el mínimo: 10
- Lista para la tarea J_2 : {23, 9, 8}. Determinamos el mínimo: 8

A partir de los datos anteriores, el criterio de selección de la mejor tarea escogería la tarea J_2 para ser ejecutada en primer lugar, debido a que demora menos que J_1 en todos los casos.

4.3.2. Ejemplo de aplicación del criterio voraz para la selección mejor

¹¹ : La formación de las listas no es tan trivial como la simple copia directa de los datos de la matriz T. Más adelante se explica el proceso correspondiente.

recurso ejecutor

Por su parte, el criterio voraz de selección del mejor recurso ejecutor será el de aquel que ejecuta la tarea correspondiente en el menor tiempo. Luego, continuando con el ejemplo anterior, para J_2 tenemos que el menor tiempo de ejecución es 8 unidades, correspondiente al recurso M_3 . Si suponemos que estamos empezando la programación de línea de producción y que $A = \{0, 0, 0\}$ procedemos a hallar

$$A_k = \min\{A_i + T_{2j} + U_2\} = \{0+23+0, 0+9+0, 0+8+0\} = 8.$$

Por lo tanto A_3 es la menor entrada y la máquina ejecutora será M3.

4.3.3. Presentación del Algoritmo Voraz propuesto

Una vez expuestos los dos criterios, a continuación se muestra el algoritmo voraz [ICIL05] el cual está dividido en las siguientes partes:

- Formación de la instancia de trabajo: ingreso de los valores de N, M, matriz T, predecesoras por tarea, inicialización de A.
- Determinación de las candidatas a ser programadas: elección de la mejor tarea y el mejor recurso ejecutor.
- Programación del resto de tareas.
- Determinación del makespan.

```

Inicio Algoritmo Voraz (N,M, T, A, S, U, V, J1, J2, ..., Jn)
1. Definir_Estructuras (N,M, T, A, S, U, V, J1, J2, ..., Jn)
2. E = ∅
3. Mientras |E| ≠ N hacer
    Inicio
    3.1 C = ∅
    3.2 Para i = 1 a N, hacer
        Si (Pi ⊆ E) ∨ (Ji ∈ E) ⇒ C = C ∪ {Ji}
        Mínimo = +∞
    3.3 Para cada Ji ∈ C, hacer
        Inicio
        3.3.1 Fi = maxj∈Ji {Uj}
        3.3.2 Si (Minj∈Ji {Tij + max(Aj, Fi)} < Mínimo ⇒
            Inicio
            3.3.2a =
            3.3.2b = ArgMinj∈Ji {Tij + max(Aj, Fi)}
        Fin Si 3.3.2
    Fin Para
    3.4 E = E ∪ {Ji}
    3.5 E = E ∪ {Ji}
    3.6 Ai = Tij + max(Aj, Fi)
    3.7 Ei = Ai
    Fin Mientras
4. Retornar makespan = maxi=1..n Ai ∪ V ∪ {LM}
Fin Algoritmo Voraz.
    
```

Figura 4-3: Algoritmo voraz propuesto

4.3.4. Comentarios sobre el algoritmo presentado

- Línea 1: lectura e inicialización de las estructuras y datos necesarios
- Línea 2: se inicializa el conjunto E en vacío porque ninguna tarea ha sido programada aún.
- Línea 3: el proceso de programación terminará cuando en el conjunto E estén contenidas las N tareas (cardinalidad de E igual a N)
 - Línea 3.1: el conjunto C de tareas candidatas a ser programadas (aptas) se inicia en vacío para cada iteración.
 - Línea 3.2: se procede a determinar las tareas candidatas. Sabemos que una tarea J_ℓ formará parte de C si sus predecesoras ya han sido ejecutadas (incluidas en E) o no presenta predecesoras (vacío incluido en E) y si además dicha J_ℓ no ha sido programada previamente (no pertenece a E).
 - Línea 3.3: Para las tareas candidatas de C seleccionaremos la mejor, así como al mejor recurso ejecutor siguiendo los criterios explicados en apartados anteriores.
 - Línea 3.3.1: Actualizamos al vector V. Escogemos la predecesora de la tarea J_ℓ que finaliza más tarde (mayor entrada de U de las predecesoras de J_ℓ)
 - Línea 3.3.2: Aplicamos el criterio de selección de la mejor tarea: para cada tarea J_ℓ en C, a cada entrada A_{ij} le sumamos el valor de V_i , o sea el tiempo de finalización de la última predecesora de J_ℓ y le sumamos también el tiempo de ejecución T_{ij} . Buscamos el mínimo valor.
 - Línea 3.3.3: Ubicamos la mejor tarea en la variable k y la mejor máquina como a aquella que ejecute a la tarea k en el menor tiempo,
 - Línea 3.4: Actualizamos S para la entrada S_i pues ahora el recurso M_i ejecuta a la tarea J_k
 - Línea 3.5: Actualizamos E con J_k porque ya está programada.
 - Línea 3.6: Actualizamos al vector A_i
- Línea 4: se calcula el makespan, que es la mayor entrada del vector A, y se retornan las tareas asignadas a cada recurso.

4.3.5. Ejemplo de aplicación del algoritmo voraz

En este apartado vamos a presentar una aplicación práctica del algoritmo anterior. Para eso trabajaremos con una instancia de tamaño 24: 8 tareas y 3 recursos. Tenemos como entrada igualmente una matriz de tiempos de ejecución $T_{8 \times 3}$ y la lista de tareas predecesoras por cada una de las tareas del proyecto. La siguiente matriz de tiempos de ejecución T y la lista de predecesoras:

A inicial	Act 10, 20, 30	A inicial	Act 10, 20, 30
B	{21, 23, 27}	B	{21, 23, 27}
J1	10, 10, 10, 10, 10, 10	J1	10, 10, 10, 10, 10, 10
J4	30, 30, 30, 30, 30, 30	J4	30, 30, 30, 30, 30, 30
Candidatos C	{21}	Candidatos C	{21}
WT	21	WT	21
Para J1	$21 - \max(A1, 21) + 10 = 10$ $21 - \max(A4, 21) + 10 = 10$ $21 - \max(A7, 21) + 10 = 10$	Para J4	$21 - \max(A1, 21) + 10 = 10$ $21 - \max(A4, 21) + 10 = 10$ $21 - \max(A7, 21) + 10 = 10$
Máximo = 10		Máximo = 10	
Mejor tarea	21	Mejor tarea	21
Mejor Máquina	M2	Mejor Máquina	M2
ti	10, 20	ti	10, 20

Tabla 4-5: Tercera y cuarta iteración voraz

La tarea J7 tiene dos predecesoras: J1 y J4; debemos considerar programarla después de la predecesora que termina más tarde. Analizando tenemos que J1 acababa en el tiempo 10 y J4 en el tiempo 30, por ende consideraremos programarla después del tiempo 30. Luego $B = \{30+27, 30+21, 30+23\} = \{57, 51, 53\}$ y elegimos M2, para posteriormente actualizar $A = \{30, 51, 20\}$

A inicial	Act 10, 20, 30	A inicial	Act 10, 20, 30
B	{21, 23, 27}	B	{21, 23, 27}
J1	10, 10, 10, 10, 10, 10	J1	10, 10, 10, 10, 10, 10
J4	30, 30, 30, 30, 30, 30	J4	30, 30, 30, 30, 30, 30
Candidatos C	{21}	Candidatos C	{21}
WT	21	WT	21
Para J1	$21 - \max(A1, 21) + 10 = 10$ $21 - \max(A4, 21) + 10 = 10$ $21 - \max(A7, 21) + 10 = 10$	Para J4	$21 - \max(A1, 21) + 10 = 10$ $21 - \max(A4, 21) + 10 = 10$ $21 - \max(A7, 21) + 10 = 10$
Máximo = 10		Máximo = 10	
Mejor tarea	21	Mejor tarea	21
Mejor Máquina	M2	Mejor Máquina	M2
ti	10, 20	ti	10, 20

Tabla 4-6: Quinta y sexta iteración voraz

Las posibles tareas a programar son J2 y J5 debido a que sus predecesoras (J1, J4 y J7 en el caso de J2 y J7 y J8 en el caso de J5) ya fueron programadas previamente. De la misma manera que cuando se enfrentaron J1 y J6, determinamos las listas B para cada una de ellas: para J2 $B = \{62, 64, 70\}$ con un mínimo de 62 y para J5 $B = \{60, 61, 71\}$ con un mínimo de 60. Por el criterio de selección de la mejor tarea elegimos J5.

A inicial	Act 10, 20, 30	A inicial	Act 10, 20, 30
B	{21, 23, 27}	B	{21, 23, 27}
J1	10, 10, 10, 10, 10, 10	J1	10, 10, 10, 10, 10, 10
J4	30, 30, 30, 30, 30, 30	J4	30, 30, 30, 30, 30, 30
Candidatos C	{21}	Candidatos C	{21}
WT	21	WT	21
Para J1	$21 - \max(A1, 21) + 10 = 10$ $21 - \max(A4, 21) + 10 = 10$ $21 - \max(A7, 21) + 10 = 10$	Para J4	$21 - \max(A1, 21) + 10 = 10$ $21 - \max(A4, 21) + 10 = 10$ $21 - \max(A7, 21) + 10 = 10$
Máximo = 10		Máximo = 10	
Mejor tarea	21	Mejor tarea	21
Mejor Máquina	M2	Mejor Máquina	M2
ti	10, 20	ti	10, 20

Tabla 4-7: Séptima y octava iteración voraz

Luego tenemos como makespan: el tiempo total de trabajo del lote es 83. Esto no significa que todos los recursos trabajaron continuamente 83 unidades de tiempo sino que la última tarea se terminó de ejecutar en el tiempo 83. Veamos el cuadro de asignaciones recursos – tareas.

Tabla 4-8: Resultados del ejemplo de aplicación del algoritmo voraz

Acumulado final	A = {60,64,83}
Makespan	83
M1	{J1,J4,J5}
M2	{J2,J7}
M3	{J3,J6,J8}

De otro lado, si utilizamos únicamente el criterio voraz para la selección del mejor recurso ejecutor sin incidir en escoger a la mejor tarea el resultado de la aplicación del algoritmo a la instancia del ejemplo anterior variaría empeorando el resultado. Veamos resumidamente la programación de cada tarea:

- Primera tarea: J1 (las estamos tomando en el orden de aparición) $A = \{10,0,0\}$
- Segunda tarea: J6. $A = \{10,0,8\}$
- Tercera tarea: J8. $A = \{10,0,20\}$
- Cuarta tarea: J4. $A = \{30,0,20\}$
- Quinta tarea: J7. $A = \{30,51,20\}$
- Sexta tarea: J2 (en el orden de aparición J2 estaría antes de J5) $A = \{62,51,20\}$
- Séptima tarea: J5 $A = \{62,61,20\}$
- Octava tarea: J3 $A = \{62,61,84\}$

En donde se nota que el makespan se incremento en 1 (de 83 a 84). Esto significa que el doble criterio de selección permitió mejorar en un 2% la simple aplicación del criterio voraz¹²

4.4. Algoritmo GRASP propuesto: lineamientos generales

De manera similar, vamos a plantear los dos criterios de selección durante el desarrollo del algoritmo GRASP:

- Un criterio GRASP de selección aleatoria de la mejor tarea a programarse, empleando la constante de relajación α .
- Un criterio de selección aleatoria del mejor recurso disponible que ejecutará la tarea seleccionada antes, empleando un parámetro θ adicional.

Se ha demostrado que tras la ejecución de numerosas iteraciones de la fase de construcción GRASP se obtienen mejores resultados que los arrojados por los algoritmos

¹² : Se ahondará sobre la eficiencia de los algoritmos en el capítulo de los experimentos numéricos.

heurísticos, dado que la aleatoriedad de las selecciones genera un espectro de soluciones mayor, en donde puede encontrarse la solución óptima. Este universo de soluciones crecerá aún más con la presencia de los dos criterios planteados.

4.4.1. Criterio GRASP para la selección aleatoria de la mejor tarea a ejecutar.

Este criterio se basa en los mismos principios que el algoritmo voraz presentado antes:

- Identificar las tareas aptas para ser programadas: es decir, las que no han sido programadas aún y sus predecesoras ya han sido ejecutadas (o no presentan predecesoras).
- Para cada una de las tareas aptas, generar la misma lista que en el algoritmo voraz: establecer los tiempos de ejecución acumulado a partir de la finalización de la última tarea predecesora que se ejecutó.
- Obtener el menor elemento de cada lista y almacenarlos en otra lista de mínimos locales. De esta nueva lista de mínimos locales seleccionaremos: el máximo y el mínimo valor las variables: peor y mejor respectivamente.
- Formaremos la lista de tareas candidatas RCL analizando cada entrada de la lista de mínimos locales: si la entrada correspondiente cumple con estar dentro del intervalo [mejor, mejor + α^* (peor-mejor)], entonces pasa a formar parte de RCL.
- Se escoge una tarea al azar de las que constituyen RCL.

4.4.2. Criterio GRASP para la selección aleatoria del mejor recurso ejecutor.

Una vez seleccionada una tarea desde RCL, se busca el mejor recurso disponible que la pueda ejecutar. Los pasos a seguir son los siguientes:

- Se forma nuevamente el vector de tiempos acumulados en función de las predecesoras de la tarea j-ésima que está siendo objeto de análisis.
- Se encuentran los valores máximos y mínimos de ejecución (tiempos de ejecución) en las variables: peor y mejor respectivamente.
- Procedemos a formar la lista de recursos candidatas MCL: todo recurso que ejecute a la tarea j-ésima en un tiempo que se encuentre en el intervalo [mejor, mejor + θ^* (peor-mejor)] forma parte de MCL.
- Igualmente seleccionamos al azar uno de ellos, que será el ejecutor de la tarea j-ésima.

4.4.3. Presentación del Algoritmo GRASP – fase de Construcción

El algoritmo [CLEI04] es el siguiente:

Figura 4-25: Algoritmo GRASP Construcción.

4.4.4. Comentarios sobre el algoritmo GRASP – fase Construcción presentado

- Línea 6: al igual que en el algoritmo voraz, el proceso acaba cuando en el conjunto E se encuentren todas las tareas (E conjunto de las tareas programadas)
 - Línea 6.1: se inicializa en vacío la lista de tareas aptas C
 - Línea 6.2: se inicializan las variables mejor y peor con las que se trabajarán los intervalos de los criterios de relajación.
 - Línea 6.4: se forma la lista de tareas aptas C
 - Línea 6.5: se inicializa en vacío la lista de mínimos locales $B_{\min}$
 - Líneas 6.6 – 6.6.2: se actualiza las entradas correspondientes de la lista V; se forma la lista $B_{\min}$ agregándole cada menor elemento de la lista de tiempos acumulados de cada tarea.
 - Líneas 6.7-6.8: se le asignan los valores máximos y mínimos de $B_{\min}$ a las variables peor y mejor respectivamente.
 - Línea 6.9: se inicializa en vacío la lista RCL.
 - Líneas 6.10-6.11: se forma la lista RCL cuando se cumpla la condición:

$$\text{condición } \min_{p \in [1, n]} \{T_p + \max\{A_p, V_i\}\} \in [\text{mejor}, \text{mejor} + \alpha * (\text{peor} - \text{mejor})]$$

luego se escoge un elemento al azar de ésta lista (tarea k)

- Líneas 6.12-6.16: se escoge el mínimo y el máximo tiempo de ejecución para la tarea escogida en 6.11. Se formará MCL a partir de los recursos que puedan ejecutar dicha tarea cumpliendo la condición:

la condición: $T_{ij} + \max(A_i, V_k) \in [\text{mejor}, \text{mejor} + \epsilon^* \text{ (peor-mejor)}]$

Finalmente también se escoge un recurso de forma aleatoria (recurso i)

[illegible][illegible]

A. Serial	Serial 1 (1)
B	Serial 2 (2)
C	Serial 3 (3)
D	Serial 4 (4)
E	Serial 5 (5)
Coordinate C	Coordinate C (1)
VF	VF (1)
Para. 17	Para. 17 (1)
Interval 102	Interval 102 (1)
102	102 (1)
Interval 103	Interval 103 (1)
103	103 (1)
Interval 104	Interval 104 (1)
104	104 (1)
Interval 105	Interval 105 (1)
105	105 (1)
Interval 106	Interval 106 (1)
106	106 (1)
Interval 107	Interval 107 (1)
107	107 (1)
Interval 108	Interval 108 (1)
108	108 (1)
Interval 109	Interval 109 (1)
109	109 (1)
Interval 110	Interval 110 (1)
110	110 (1)
Interval 111	Interval 111 (1)
111	111 (1)
Interval 112	Interval 112 (1)
112	112 (1)
Interval 113	Interval 113 (1)
113	113 (1)
Interval 114	Interval 114 (1)
114	114 (1)
Interval 115	Interval 115 (1)
115	115 (1)
Interval 116	Interval 116 (1)
116	116 (1)
Interval 117	Interval 117 (1)
117	117 (1)
Interval 118	Interval 118 (1)
118	118 (1)
Interval 119	Interval 119 (1)
119	119 (1)
Interval 120	Interval 120 (1)
120	120 (1)
Interval 121	Interval 121 (1)
121	121 (1)
Interval 122	Interval 122 (1)
122	122 (1)
Interval 123	Interval 123 (1)
123	123 (1)
Interval 124	Interval 124 (1)
124	124 (1)
Interval 125	Interval 125 (1)
125	125 (1)
Interval 126	Interval 126 (1)
126	126 (1)
Interval 127	Interval 127 (1)
127	127 (1)
Interval 128	Interval 128 (1)
128	128 (1)
Interval 129	Interval 129 (1)
129	129 (1)
Interval 130	Interval 130 (1)
130	130 (1)
Interval 131	Interval 131 (1)
131	131 (1)
Interval 132	Interval 132 (1)
132	132 (1)
Interval 133	Interval 133 (1)
133	133 (1)
Interval 134	Interval 134 (1)
134	134 (1)
Interval 135	Interval 135 (1)
135	135 (1)
Interval 136	Interval 136 (1)
136	136 (1)
Interval 137	Interval 137 (1)
137	137 (1)
Interval 138	Interval 138 (1)
138	138 (1)
Interval 139	Interval 139 (1)
139	139 (1)
Interval 140	Interval 140 (1)
140	140 (1)
Interval 141	Interval 141 (1)
141	141 (1)
Interval 142	Interval 142 (1)
142	142 (1)
Interval 143	Interval 143 (1)
143	143 (1)
Interval 144	Interval 144 (1)
144	144 (1)
Interval 145	Interval 145 (1)
145	145 (1)
Interval 146	Interval 146 (1)
146	146 (1)
Interval 147	Interval 147 (1)
147	147 (1)
Interval 148	Interval 148 (1)
148	148 (1)
Interval 149	Interval 149 (1)
149	149 (1)
Interval 150	Interval 150 (1)
150	150 (1)
Interval 151	Interval 151 (1)
151	151 (1)
Interval 152	Interval 152 (1)
152	152 (1)
Interval 153	Interval 153 (1)
153	153 (1)
Interval 154	Interval 154 (1)
154	154 (1)
Interval 155	Interval 155 (1)
155	155 (1)
Interval 156	Interval 156 (1)
156	156 (1)
Interval 157	Interval 157 (1)
157	157 (1)
Interval 158	Interval 158 (1)
158	158 (1)
Interval 159	Interval 159 (1)
159	159 (1)
Interval 160	Interval 160 (1)
160	160 (1)
Interval 161	Interval 161 (1)
161	161 (1)
Interval 162	Interval 162 (1)
162	162 (1)
Interval 163	Interval 163 (1)
163	163 (1)
Interval 164	Interval 164 (1)
164	164 (1)
Interval 165	Interval 165 (1)
165	165 (1)
Interval 166	Interval 166 (1)
166	166 (1)
Interval 167	Interval 167 (1)
167	167 (1)
Interval 168	Interval 168 (1)
168	168 (1)
Interval 169	Interval 169 (1)
169	169 (1)
Interval 170	Interval 170 (1)
170	170 (1)
Interval 171	Interval 171 (1)
171	171 (1)
Interval 172	Interval 172 (1)
172	172 (1)
Interval 173	Interval 173 (1)
173	173 (1)
Interval 174	Interval 174 (1)
174	174 (1)
Interval 175	Interval 175 (1)
175	175 (1)
Interval 176	Interval 176 (1)
176	176 (1)
Interval 177	Interval 177 (1)
177	177 (1)
Interval 178	Interval 178 (1)
178	178 (1)
Interval 179	Interval 179 (1)
179	179 (1)
Interval 180	Interval 180 (1)
180	180 (1)
Interval 181	Interval 181 (1)
181	181 (1)
Interval 182	Interval 182 (1)
182	182 (1)
Interval 183	Interval 183 (1)
183	183 (1)
Interval 184	Interval 184 (1)
184	184 (1)
Interval 185	Interval 185 (1)
185	185 (1)
Interval 186	Interval 186 (1)
186	186 (1)
Interval 187	Interval 187 (1)
187	187 (1)
Interval 188	Interval 188 (1)
188	188 (1)
Interval 189	Interval 189 (1)
189	189 (1)
Interval 190	Interval 190 (1)
190	190 (1)
Interval 191	Interval 191 (1)
191	191 (1)
Interval 192	Interval 192 (1)
192	192 (1)
Interval 193	Interval 193 (1)
193	193 (1)
Interval 194	Interval 194 (1)
194	194 (1)
Interval 195	Interval 195 (1)
195	195 (1)
Interval 196	Interval 196 (1)
196	196 (1)
Interval 197	Interval 197 (1)
197	197 (1)
Interval 198	Interval 198 (1)
198	198 (1)
Interval 199	Interval 199 (1)
199	199 (1)
Interval 200	Interval 200 (1)
200	200 (1)
Interval 201	Interval 201 (1)
201	201 (1)
Interval 202	Interval 202 (1)
202	202 (1)
Interval 203	Interval 203 (1)
203	203 (1)
Interval 204	Interval 204 (1)
204	204 (1)
Interval 205	Interval 205 (1)
205	205 (1)
Interval 206	Interval 206 (1)
206	206 (1)
Interval 207	Interval 207 (1)
207	207 (1)
Interval 208	Interval 208 (1)
208	208 (1)
Interval 209	Interval 209 (1)
209	209 (1)
Interval 210	Interval 210 (1)
210	210 (1)
Interval 211	Interval 211 (1)
211	211 (1)
Interval 212	Interval 212 (1)
212	212 (1)
Interval 213	Interval 213 (1)
213	213 (1)
Interval 214	Interval 214 (1)
214	214 (1)
Interval 215	Interval 215 (1)
215	215 (1)
Interval 216	Interval 216 (1)
216	216 (1)
Interval 217	Interval 217 (1)
217	217 (1)
Interval 218	Interval 218 (1)
218	218 (1)
Interval 219	Interval 219 (1)
219	219 (1)
Interval 220	Interval 220 (1)
220	220 (1)
Interval 221	Interval 221 (1)
221	221 (1)
Interval 222	Interval 222 (1)
222	222 (1)
Interval 223	Interval 223 (1)
223	223 (1)
Interval 224	Interval 224 (1)
224	224 (1)
Interval 225	Interval 225 (1)
225	225 (1)
Interval 226	Interval 226 (1)
226	226 (1)
Interval 227	Interval 227 (1)
227	227 (1)
Interval 228	Interval 228 (1)
228	228 (1)
Interval 229	Interval 229 (1)
229	229 (1)
Interval 230	Interval 230 (1)
230	230 (1)
Interval 231	Interval 231 (1)
231	231 (1)
Interval 232	Interval 232 (1)
232	232 (1)
Interval 233	Interval 233 (1)
233	233 (1)
Interval 234	Interval 234 (1)
234	234 (1)
Interval 235	Interval 235 (1)
235	235 (1)
Interval 236	Interval 236 (1)
236	236 (1)
Interval 237	Interval 237 (1)
237	237 (1)
Interval 238	Interval 238 (1)
238	238 (1)
Interval 239	Interval 239 (1)
239	239 (1)
Interval 240	Interval 240 (1)
240	240 (1)
Interval 241	Interval 241 (1)
241	241 (1)
Interval 242	Interval 242 (1)
242	242 (1)
Interval 243	Interval 243 (1)
243	243 (1)
Interval 244	Interval 244 (1)
244	244 (1)
Interval 245	Interval 245 (1)
245	245 (1)
Interval 246	Interval 246 (1)
246	246 (1)
Interval 247	Interval 247 (1)
247	247 (1)
Interval 248	Interval 248 (1)
248	248 (1)
Interval 249	Interval 249 (1)
249	249 (1)
Interval 250	Interval 250 (1)
250	250 (1)
Interval 251	Interval 251 (1)
251	251 (1)
Interval 252	Interval 252 (1)
252	252 (1)
Interval 253	Interval 253 (1)
253	253 (1)
Interval 254	Interval 254 (1)
254	254 (1)
Interval 255	Interval 255 (1)
255	255 (1)
Interval 256	Interval 256 (1)
256	256 (1)
Interval 257	Interval 257 (1)
257	257 (1)
Interval 258	Interval 258 (1)
258	258 (1)
Interval 259	Interval 259 (1)
259	259 (1)
Interval 260	Interval 260 (1)
260	260 (1)
Interval 261	Interval 261 (1)
261	261 (1)
Interval 262	Interval 262 (1)
262	262 (1)
Interval 263	Interval 263 (1)
263	263 (1)
Interval 264	Interval 264 (1)
264	264 (1)
Interval 265	Interval 265 (1)
265	265 (1)
Interval 266	Interval 266 (1)
266	266 (1)
Interval 267	Interval 267 (1)
267	267 (1)
Interval 268	Interval 268 (1)
268	268 (1)
Interval 269	Interval 269 (1)
269	269 (1)
Interval 270	Interval 270 (1)
270	270 (1)
Interval 271	Interval 271 (1)
271	271 (1)
Interval 272	Interval 272 (1)
272	272 (1)
Interval 273	Interval 273 (1)
273	273 (1)
Interval 274	Interval 274 (1)
274	274 (1)
Interval 275	Interval 275 (1)
275	275 (1)
Interval 276	Interval 276 (1)
276	276 (1)
Interval 277	Interval 277 (1)
277	277 (1)
Interval 278	Interval 278 (1)
278	278 (1)
Interval 279	Interval 279 (1)
279	279 (1)
Interval 280	Interval 280 (1)
280	280 (1)
Interval 281	Interval 281 (1)
281	281 (1)
Interval 282	Interval 282 (1)
282	282 (1)
Interval 283	Interval 283 (1)
283	283 (1)
Interval 284	Interval 284 (1)
284	284 (1)
Interval 285	Interval 285 (1)
285	285 (1)
Interval 286	Interval 286 (1)
286	286 (1)
Interval 287	Interval 287 (1)
287	287 (1)
Interval 288	Interval 288 (1)
288	288 (1)
Interval 289	Interval 289 (1)
289	289 (1)
Interval 290	Interval 290 (1)
290	290 (1)
Interval 291	Interval 291 (1)
291	291 (1)
Interval 292	Interval 292 (1)
292	292 (1)
Interval 293	Interval 293 (1)
293	293 (1)
Interval 294	Interval 294 (1)
294	294 (1)
Interval 295	Interval 295 (1)
295	295 (1)
Interval 296	Interval 296 (1)
296	296 (1)
Interval 297	Interval 297 (1)
297	297 (1)
Interval 298	Interval 298 (1)
298	298 (1)
Interval 299	Interval 299 (1)
299	299 (1)
Interval 300	Interval 300 (1)
300	300 (1)
Interval 301	Interval 301 (1)
301	301 (1)
Interval 302	Interval 302 (1)
302	302 (1)
Interval 303	Interval 303 (1)
303	303 (1)
Interval 304	Interval 304 (1)
304	304 (1)
Interval 305	Interval 305 (1)
305	305 (1)
Interval 306	Interval 306 (1)
306	306 (1)
Interval 307	Interval 307 (1)
307	307 (1)
Interval 308	Interval 308 (1)
308	308 (1)
Interval 309	Interval 309 (1)
309	309 (1)
Interval 310	Interval 310 (1)
310	310 (1)
Interval 311	Interval 311 (1)
311	311 (1)
Interval 312	Interval 312 (1)
312	312 (1)
Interval 313	Interval 313 (1)
313	313 (1)
Interval 314	Interval 314 (1)
314	314 (1)
Interval 315	Interval 315 (1)
315	315 (1)
Interval 316	Interval 316 (1)
316	316 (1)
Interval 317	Interval 317 (1)
317	317 (1)
Interval 318	Interval 318 (1)
318	318 (1)
Interval 319	Interval 319 (1)
319	319 (1)
Interval 320	Interval 320 (1)
320	320 (1)
Interval 321	Interval 321 (1)
321	321 (1)
Interval 322	Interval 322 (1)
322	322 (1)
Interval 323	Interval 323 (1)
323	323 (1)
Interval 324	Interval 324 (1)
324	324 (1)
Interval 325	Interval 325 (1)
325	325 (1)
Interval 326	Interval 326 (1)
326	326 (1)
Interval 327	Interval 327 (1)
327	327 (1)
Interval 328	Interval 328 (1)
328	328 (1)
Interval 329	Interval 329 (1)
329	329 (1)
Interval 330	Interval 330 (1)
330	330 (1)
Interval 331	Interval 331 (1)
331	331 (1)
Interval 332	Interval 332 (1)
332	332 (1)
Interval 333	Interval 333 (1)
333	333 (1)
Interval 334	Interval 334 (1)
334	334 (1)
Interval 335	Interval 335 (1)
335	335 (1)
Interval 336	Interval 336 (1)
336	336 (1)
Interval 337	Interval 337 (1)
337	337 (1)
Interval 338	Interval 338 (1)
338	338 (1)
Interval 339	Interval 339 (1)
339	339 (1)
Interval 340	Interval 340 (1)
340	340 (1)
Interval 341	Interval 341 (1)
341	341 (1)
Interval 342	Interval 342 (1)
342	342 (1)
Interval 343	Interval 343 (1)
343	343 (1)
Interval 344	Interval 344 (1)
344	344 (1)
Interval 345	Interval 345 (1)
345	345 (1)
Interval 346	Interval 346 (1)
346	346 (1)
Interval 347	Interval 347 (1)
347	347 (1)
Interval 348	Interval 348 (1)
348	348 (1)
Interval 349	Interval 349 (1)
349	349 (1)
Interval 350	Interval 350 (1)
350	350 (1)
Interval 351	Interval 351 (1)
351	351 (1)
Interval 352	Interval 352 (1)
352	352 (1)
Interval 353	Interval 353 (1)
353	353 (1)
Interval 354	Interval 354 (1)
354	354 (1)
Interval 355	Interval 355 (1)
355	355 (1)
Interval 356	Interval 356 (1)

Tabla 4□12: Iteraciones 7 y 8 del algoritmo GRASP

Se está manejando dos tipos de criterios en el algoritmo GRASP para elegir la tarea a realizar y el recurso destinado a hacerla.

El proceso de elegir el recurso ejecutor más apto parte de la disponibilidad del recurso (está libre o no) y la capacidad que tenga de poder realizar la tarea que se le pretende asignar. Este tema dentro de los proyectos es muy importante, porque los distintos actores-recursos siguen roles distintos, como se ha venido mencionado a lo largo de los anteriores capítulos y es probable que disponibilidad no esté asociada directamente con capacidad de ejecución: por ejemplo, puede estar disponible un recurso programador cuando la tarea que se desea asignar no sea de la fase de programación (usándose RUP o no). Este panorama le añade complejidad al problema.

En resumen, el algoritmo GRASP planteado permite considerar estas situaciones propias de un equipo de desarrollo de software sin perder generalidad con que se está programando una *línea de producción – de software*.

Capítulo V. Implementación de los algoritmos

5.1. Modelado de negocios del prototipo

El modelado de negocios presentará el panorama de acción de los diferentes roles involucrados en un proyecto de software, teniendo como actor principal al planificador o administrador del proyecto, que procederá a realizar la determinación de las tareas y la gestión de los recursos correspondientes:

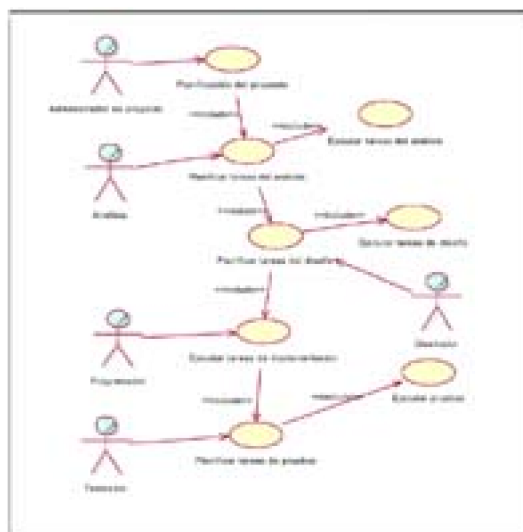


Figura 5-1: Modelado de negocios del escenario de planificación –ejecución de tareas

Cualquiera de los roles (actores) tienen o que planificar las tareas y decidir sobre ellas o tiene que ejecutarlas empleando determinados recursos; en este punto, no solamente es el trabajador visto como recurso máquina sino también el resto de elementos de tecnología que necesite.

5.2. Diagrama de actividades

El proceso de planificación, asignación de tareas y ejecución de operaciones para generar ciertos artefactos (prototipos) por parte de los recursos involucrados en un proyecto sigue una secuencia, sea empleado RUP o siguiendo otra metodología. En el siguiente diagrama de actividades se puede observar esta secuencia:



Figura 5-1: Diagrama de actividades

Claramente se puede apreciar el orden con que los procesos de negocio (workflow) un paso en el flujo de trabajo o la ejecución de una operación, requiere que previamente se haya ejecutado de forma completa, el resto de actividades. El gráfico describe además la concurrencia de los procesos de ingreso de datos (tareas y recursos) y la

Un diagrama de actividades es provechoso para entender el comportamiento de alto nivel de la ejecución del sistema, sin profundizar en los detalles internos de los mensajes entre los componentes que se van a implementar en base a los algoritmos propuestos.

Siguiendo el modelo de Larman [La99] los esquemas siguientes mostrarán las especificaciones de los casos de uso de sistema, una vez comprendida la lógica del negocio en lo concerniente a esta etapa de planificación previa (de las fases iniciales RUP y referida también a los refinamientos sucesivos que sufren los planes de dichas fases)

Tabla 5□1: Especificación Ingreso de los datos de entrada

"Programa Cybertesis PERÚ - Derechos son del Autor"	71
---	----

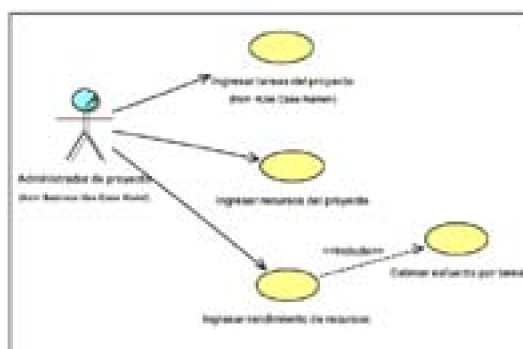


Figura 5□2: Diagrama de caso de uso Ingreso de los datos de entrada

5.3.2. Especificación: Caso de uso Ejecutar programación

Caso de uso de sistema: Ejecutar programación		
Actores	Administrador del sistema	
Precondición	Ingresar parámetros para el algoritmo GRASP (el nombre de la instancia, etc.).	
Procedimiento	El administrador de proyectos ingresará al sistema los datos requeridos para la ejecución del algoritmo GRASP o luego después de proceso. Los resultados se almacenarán en dispositivos de almacenamiento.	
Recurso	Proceso de ejecución de sistema	
Secuencia	Acción de los actores	Respuesta del sistema
1	El Administrador de proyectos ingresará al sistema los datos requeridos para la ejecución del algoritmo GRASP.	El sistema mostrará los datos ingresados al sistema y los datos requeridos para la ejecución del algoritmo GRASP.
2	El Administrador de proyectos ingresará al sistema los datos requeridos para la ejecución del algoritmo GRASP.	El sistema mostrará el algoritmo GRASP y los datos requeridos para la ejecución del algoritmo GRASP.
3		El sistema mostrará el algoritmo GRASP y los datos requeridos para la ejecución del algoritmo GRASP.
Fin del caso de uso		
4.	El administrador de proyectos ingresará al sistema los datos requeridos para la ejecución del algoritmo GRASP.	El sistema mostrará los datos ingresados al sistema y los datos requeridos para la ejecución del algoritmo GRASP.

Tabla 5□2 Especificación Ejecutar programación

El diagrama de caso de uso que le corresponde es:

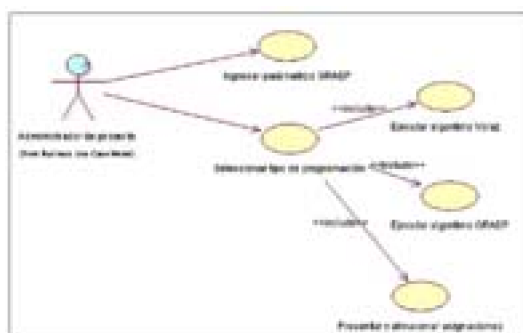


Figura 5□3: Diagrama de caso de uso Ejecutar programación

5.4. Diagramas de secuencia

5.4.1. Diagrama de secuencia Ingresar datos de entrada

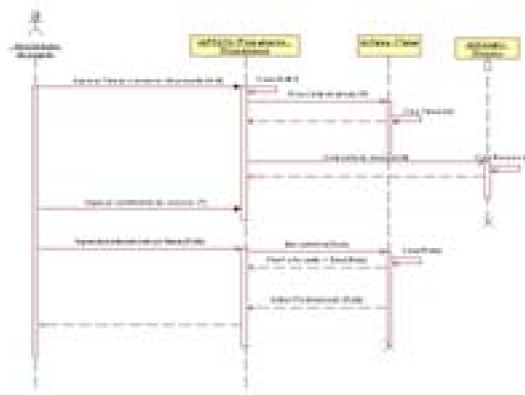


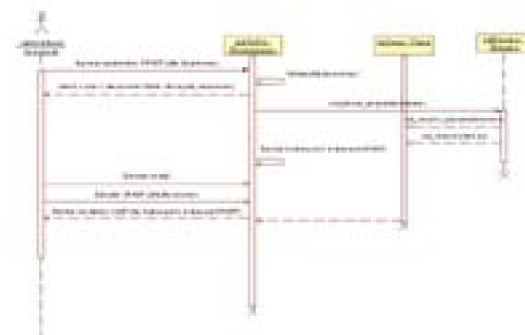
Figura 5□4: Diagrama de secuencia Ingresar datos de entrada

En este diagrama de secuencia se muestra como el sistema va realizando las validaciones para crear un objeto de la clase TPROGRAMACION¹³, el cual necesita de los datos de las tareas, los recursos, las performances y restricciones de los recursos y la lista de las predecesoras. Precisamente, se van creando objetos de las clases TTarea y TRecurso para la inicialización del objeto objPROTA. La lista de predecesoras por tarea, por su parte, no es más que un proceso de carga de un archivo (como se verá más adelante).

Luego, en caso en que se ha hecho de forma correcta la carga de los datos del archivo dada la ruta respectiva, se procede a crear definitivamente el objeto objPROTA.

5.4.2. Diagrama de secuencia Ejecutar programación

Este diagrama muestra las validaciones que el sistema hace cuando el Administrador de proyectos solicita la ejecución de uno de los algoritmos. Las validaciones son extremadamente importantes en la aplicación del algoritmo GRASP pues valores erróneos de α , θ y la cantidad de iteraciones produciría un makespan deficiente en comparación con los resultados voraces.



¹³ : Ver detalle en el diagrama de clases.

Figura 5-6: Diagrama de secuencia Ejecutar programación

Adicionalmente el sistema contemplará el proceso de calibración de los parámetros requeridos por el algoritmo GRASP de forma automática, proceso que no está siendo considerado en los diagramas de secuencia.

5.5. Diagrama de clases

El diagrama de clases mostrará tres clases:

- La clase TTarea que incluye toda la información relevante a una tarea del proyecto, incluyendo la lista de predecesoras (relación recursiva) y el recurso ejecutor.
- La clase TRecurso que incluye toda la información de los recursos junto con la lista de tareas asignadas a él, para su ejecución.
- La clase TProgramacion que junta los datos de todas las tareas y los recursos con que cuenta el proyecto. Manejará además la información necesaria para la ejecución de los algoritmos propuestos.



Figura 5-7: Diagrama de clases

Se han omitido los atributos y las operaciones de la clase TProgramacion porque se hará referencia a ellos en siguientes apartados.

5.6. Consideraciones sobre el ambiente de desarrollo

Para el diseño de la solución se consideró emplear la metodología orientada a objetos, principalmente para la generación de un sistema visual que permita tanto la ejecución particular de instancias de prueba como para la ejecución de los experimentos numéricos a los que fueron sujetos los algoritmos, sobre los que tratará el siguiente capítulo.

5.6.1. Entorno de desarrollo utilizado

Para trabajar en entorno Windows se empleó el entorno de desarrollo **Borland DELPHI**

5.0 Enterprise (Build 5.62) sobre una plataforma de sistema operativo Microsoft Windows 2000 (Service Pack 3).

El motivo de la elección de la herramienta radicó en ser uno de los únicos lenguajes de programación (Object Pascal) puramente orientado a objetos como Java, C++, etc. Dado que se quiso desarrollar una interfase visual amigable y de fácil manejo por parte del usuario final, se prefirió un entorno visual a uno DOS como lo podría ser C++.

DELPHI facilita la definición de numerosas clases y la modularización en unidades sin que su número afecte el tamaño final del ejecutable generado, que es más pequeño que cualquier generado por otro entorno visual como JBuilder, Visual Basic, etc. En sí, se trató de aprovechar las características RAD (del inglés rapid application development) o desarrollo rápido de un sistema en entorno Windows, que presenta DELPHI.

5.6.2. Dispositivos de almacenamiento de datos: archivos usados.

Como se ha visto en los apartados referentes a los algoritmos en sí, se requieren ciertos dispositivos que almacenen la información básica para su ejecución. El sistema implementado empleará *archivos de texto* para el trabajo con información masiva.

De la misma forma, se podría efectuar las modificaciones del caso para que el sistema cargue a través de manejadores de base de datos de escritorio (Microsoft Access, tablas de FOX, de DBASE, de PARADOX, etc.), los datos necesarios: entre éstos encontraríamos a la matriz de tiempos de ejecución y la lista de predecesoras.

5.6.3. Alcances y limitaciones del prototipo implementado

- El prototipo considera los tiempos de ejecución de las tareas en unidades genéricas de tiempo, en donde los valores no pasan de 100 unidades de tiempo.
- El prototipo presenta un tope máximo de iteraciones para generar etapas GRASP construcción, si bien este límite es parametrizable. Si dentro de esos ciclos no logra superar a los resultados del algoritmo voraz, el proceso se detendrá sin ejecutar la fase de mejoría.
- Para cualquier instancia, el máximo de iteraciones de la fase de construcción será 2500¹⁴

5.6.4. Módulos del sistema y clases más importantes

A continuación se listan los códigos fuente de implementación de dos de las clases principales del sistema, Primero la clase TTarea:

¹⁴ : Aclaremos que esta situación probablemente se deba a la configuración de hardware donde se realizaron los experimentos. Es sostenible, que en mejores configuraciones se pueda exceder dicho valor.

"Programa Cybertesis PERÚ - Derechos son del Autor"

- En el bloque *private* de la línea 1, se encuentran los elementos básicos de la instancia: cantidad de tareas y recursos, así como el makespan resultado de la aplicación de cualquiera de los algoritmos.

```

TTarea = class
private
  F1, id Integer
  F1, ejemplar Integer
  F1, duracion Integer
  F1, ejemplar Integer
  procedure Set1, id Integer Value Integer;
  procedure Set1, ejemplar Integer Value Integer;
  procedure Set1, duracion Integer Value Integer;
  procedure Set1, ejemplar Integer Value Integer;
public
  In, programa Boolean;
  In, SinPredecesores Boolean;
  Predecesores TVecPredecesores; [Lista de predecesores de la tarea]
  i, NumPre Integer; [numero de predecesores]
  In, Predecesores [predecesores boolean]
  constructor Create1, id Integer Integer;
  destructor Destroy1;
  procedure Print, SetarPredecesores List TList Integer;
  procedure Print, SetarSinPredecesores Sin, Boolean;
  procedure Print, i Integer Integer TTarea;
published
  property 1, id Integer read F1, id write Set1, id; [id de la tarea]
  property 1, ejemplar Integer read F1, ejemplar write Set1, ejemplar;
  property 1, duracion Integer read F1, duracion write Set1, duracion;
  property 1, ejemplar Integer read F1, ejemplar write Set1, ejemplar;
end

```

- En las líneas 2-2.4 tenemos las principales estructuras de datos manejadas: lista de tareas, matriz de tiempos de ejecución, vector de tiempos acumulados, etc.
- Entre las líneas 3-3.7 están los procesos comunes a los dos algoritmos; algunos de ellos son: verificación de las predecesoras ejecutadas, determinación de la última predecesora en ejecutarse, etc.
- En la línea 4 se encuentra el algoritmo voraz.
- Entre la línea 5-5.3 tenemos los procesos involucrados en el algoritmo GRASP

Otros tipos de datos adicionales son los siguientes:

```

TVecPredecesoras = array [1..MAX_TAREAS] of integer;
TVecAcumulados = array [1..MAX_MAQ] of integer;
TVecTiemposMaquina = array [1..MAX_TAREAS, 1..MAX_MAQ]
of integer;
TVecTareas = array [1..MAX_TAREAS] of TArea;
    
```

Figura 5□3: Otras estructuras de datos

En donde:

- TVecPredecesoras: lista de predecesoras por tarea (una por tarea)
- TVecAcumulados: lista de tiempos acumulados por recurso.
- TVecTiemposxMaquina: Matriz de tiempos de ejecución de orden NxM.
- TVectTareas: lista de todas las tareas del lote.

5.7. Módulos del sistema

En la siguiente figura presentamos la estructura general del sistema:

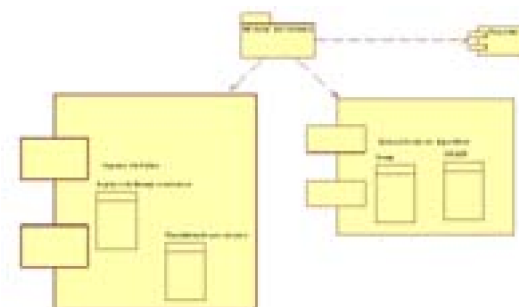


Figura 5□4 : Estructura del sistema

La estructura muestra una división en módulos natural, en donde se están separando los procesos de lectura de datos de la ejecución de los algoritmos y de los reportes. Se presentará también los reportes mostrando la información de la performance de los algoritmos.

5.7.1. Modulo de ingreso de datos de la instancia de trabajo

El módulo de ingreso de datos permite la configuración de la siguiente información:

- Información general:
 - Cantidad de tareas
 - Cantidad de recursos
 - Ruta del archivo conteniendo los valores de la matriz de rendimientos por recurso
 - Ruta del archivo conteniendo la lista de predecesoras por tarea
- Para la ejecución del algoritmo GRASP:
 - Cantidad de iteraciones que se realizará la fase de construcción GRASP
 - Valor de la constante de relajación α
 - Valor de la constante de relajación θ

En la siguiente figura se ven las dos formas de realizar la carga de una instancia por medio del sistema: de forma automatizada (indicando ruta de los archivos con data) o de forma manual (ingresando una a una las predecesoras por tarea):



Figura 5□5: Pantalla de ingreso de datos

5.7.1.1. Ingreso de datos a través de archivos: validaciones.

Este ingreso automatizado permite indicarle al sistema, la ruta donde se encuentra el archivo que contiene la lista de predecesoras por tarea y la matriz de rendimientos. Ambos archivos son de tipo texto (extensión txt) y deben cumplir ciertas características, para lo cual que se procede a realizar las siguientes validaciones internas:

- Una vez que el usuario ingresa la ruta del archivo que representa a la *matriz de tiempos de ejecución*, verifica que el orden sea NxM o sea que existan NxM datos dentro del archivo de texto correspondiente.
- Una vez que el usuario ingresa la ruta del archivo que representa la lista de predecesoras, el sistema verifica su existencia.

En ambos casos emitirá sendos mensajes de error en donde le informa al usuario de la circunstancia producida. Para la lista de predecesoras se considera el siguiente formato adecuado: se está considerando un archivo de texto en donde cada línea liste los identificadores de sus predecesoras; en caso de no tenerlas, deberá aparecer -1 como valor arbitrario. Un modelo de archivo de predecesoras se puede observar en la siguiente figura:

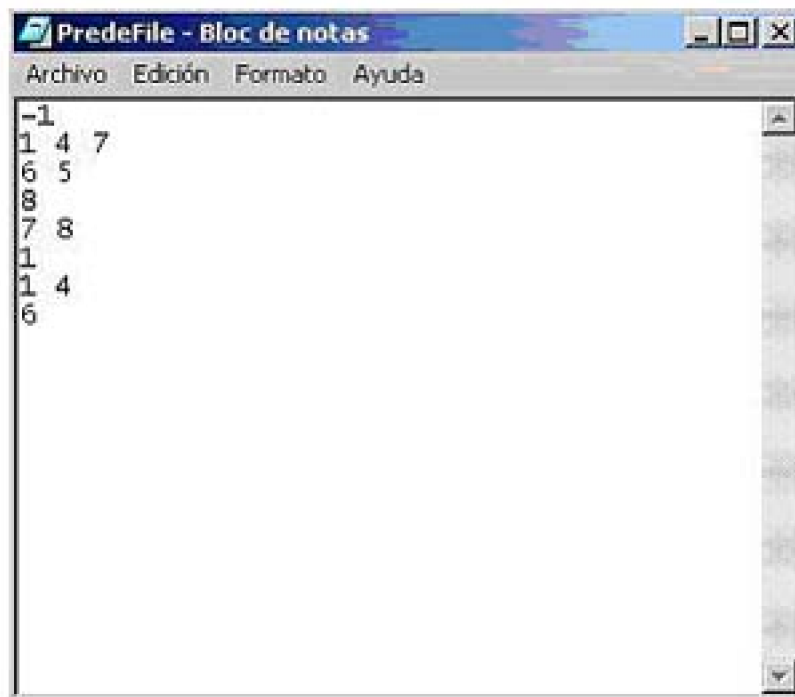


Figura 5□6: Formato de un archivo de predecesoras por tarea

Nótese que la primera línea, que correspondería a la lista de predecesoras de la primera tarea, contiene -1 lo que significa que *carece de predecesoras*. Para el resto de tareas, cada línea corresponde a las predecesoras que le corresponde no importando el orden de aparición. Así por ejemplo:

- La tarea 2 tiene como predecesoras a las tareas 1, 4 y 7.
- La tarea 3 tiene como predecesoras a las tareas 6 y 5 (desorden)

5.7.1.2. Ingreso manual de la lista de predecesoras

La otra posibilidad que ofrece el sistema es el ingreso manual de la lista de predecesoras. Para esto, la opción de carga de los archivos de predecesoras (cuadro de diálogo para abrir archivos) estará deshabilitado. El sistema sigue los siguientes pasos:

- Una vez que el usuario ingresa los valores de N y M respectivamente, en el menú correspondiente a *predecesoras* debe seleccionar la opción de ingreso manual.
- En el lado derecho aparecerán todos los identificadores (un número o código¹⁵) de las N tareas, en una lista.

- Si el usuario selecciona una de las tareas en particular, procederá a ingresar las predecesoras de dicha tarea.
- La presencia o no de predecesoras se determina seleccionando una de las dos opciones de la parte inferior derecha.



Figura 5□7: Ingreso manual de predecesoras por tarea

- En caso tenga predecesoras, aparecen en el cuadro de lista inferior N-1 tareas (una tarea no es predecesora de sí misma por eso se la omite) y el usuario selecciona las tareas que crea conveniente (dando un check a los cuadros adjuntos).
- Después de seleccionar las predecesoras, el botón grabar va formando internamente el archivo de texto con esta información.
- El proceso debe repetirse para todas las tareas.



Figura 5□8: Ingreso manual de la información de las tareas (predecesoras)

5.7.2. Módulo de ejecución de los algoritmos

Una vez terminada la carga de los datos, y realizada las validaciones internas necesarias se procede a la ejecución de los algoritmos siguiendo el flujo:

¹⁵ : Este atributo de la tarea puede cambiarse por ejemplo, con una breve descripción del proceso que representa. Para efectos de demostración algorítmica, hemos simplificado el dato a un número entero.

- Se forma un objeto con los datos cargados anteriormente (N, M, T)
- Se ejecutan tanto el algoritmo voraz como las sucesivas iteraciones de la GRASP construcción
- Se presentan los makespan correspondiente a cada una de las aplicaciones de los algoritmos.

Por medio de simples mensajes, el usuario es advertido del resultado de la aplicación de cada algoritmo. Esto se aprecia en la figura a continuación:



Figura 5□9: Presentación del makespan del algoritmo GRASP

5.7.3. Módulo de reportes y presentación de resultados

El sistema presenta tanto para el caso del algoritmo voraz como para el GRASP la siguiente información:

- Asignaciones de tareas por recurso
- Tiempo ocioso de cada recurso: comparado con el tiempo efectivo de trabajo.
- Performance general de los recurso: donde se muestra gráficamente
 - Tiempo ocioso y efectivo de trabajo de toda la línea
 - Tiempo ocioso comparado por máquina

De la misma forma se presenta información comparativa entre los dos algoritmos de la siguiente manera:

- Makespan
- Recurso con el mayor y menor tiempo acumulado de proceso
- Recurso que ejecuta la mayor y menor cantidad de tareas
- Recurso con el mayor y menor tiempo ocioso acumulado.
- Porcentaje de mejora del algoritmo GRASP frente al algoritmo voraz.

Algunas de las pantallas que muestran toda esta información son las siguientes:



Figura 5□10: Pantalla general de los reportes

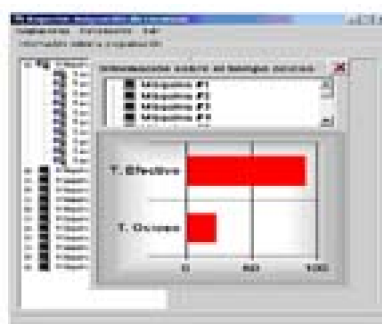


Figura 5□11: Asignaciones tareas por máquina y tiempo ocioso/efectivo por máquina

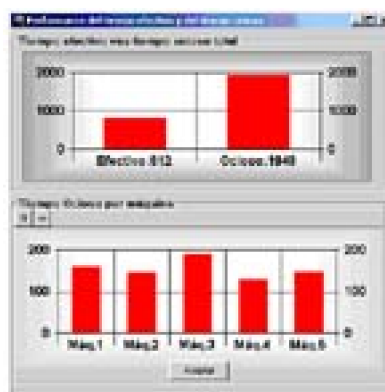


Figura 5□12: Tiempo efectivo y ocioso total. Tiempo ocioso por máquina (gráfico)



Figura 5□13: Máximos y mínimos de la programación de tareas

5.8. Requerimientos mínimos de hardware y software

A la luz de las pruebas se necesita un equipo con las siguientes características para un funcionamiento adecuado del sistema implementado:

5.8.1 Configuración de hardware mínimo

- Procesador Intel o compatible.
- Velocidad: igual o superior a 200 MHz.
- Espacio libre en disco duro: 2Mb.
- Mouse, teclado.

5.8.2. Configuración de software mínimo.

- Sistema operativo Windows 3.X, 9X, NT, ME, 2000, XP.
- Borland Delphi 5.0 o superior
- Presencia de la librería dinámica de enlace qtintf70.dll

Capítulo VI. Experimentos numéricos

6.1. Preliminares

En este capítulo se toman en cuenta las consideraciones realizadas durante los experimentos numéricos a los que fue sometido el prototipo que implementaba los algoritmos propuestos:

- Las configuraciones de hardware y de software en las que se llevaron a cabo las pruebas del sistema que implementaba los algoritmos propuestos.
- El desempeño del sistema implementado.
- La naturaleza de las instancias utilizadas para los experimentos.
- El proceso de calibración de las constantes de relajación GRASP empleadas en los experimentos: valores de α y θ .
- Los resultados obtenidos en los experimentos: analizaremos la calidad de las soluciones voraz y GRASP.
- Los resultados de las pruebas exactas efectuadas sobre el modelo matemático del problema, hechas en programas que resuelven modelos de programación no lineal (uso del LINDO)

- Análisis de la calidad real de las soluciones GRASP frente a los resultados exactos a partir del análisis de factores tales como la cantidad de iteraciones realizadas durante la fase de construcción GRASP y los valores de la constante de relajación α empleados.
- Análisis de la calidad de los algoritmos desde la perspectiva del desarrollo de proyectos de Ingeniería de Software.

6.2. Hardware y software empleados

La siguiente es la configuración de hardware del equipo empleado para la realización de las pruebas:

- Procesador: Genuine Intel® familia 6 modelo 7
- Velocidad: 1.7 GHz – Pentium IV.
- Memoria: 256 Mb de memoria RAM

De otro lado, para las pruebas en entorno WINDOWS se empleó MS Windows 2000, service pack 4. Adicionalmente se han realizado pruebas en MS Windows XP, service pack 2, para las instancias más grandes.

6.3. Los problemas test: proporción tareas / recursos

En la literatura no existen valores de prueba ni para la cantidad de tareas o máquinas para las variantes de índole industrial. Para esta variante en particular del task scheduling, tampoco. Para el problema del flow scheduling por ejemplo, Feo [FV91] presenta instancias con proporciones entre las tareas y las máquinas de 5 a 1 (en el JSP se descompone hasta un total de 32 operaciones por tarea)

Basados en estas proporciones y sobre la investigación realizada en trabajos anteriores del autor, se ha considerado como una proporción razonable entre las tareas y los recursos (para poder observar claramente los efectos del algoritmo GRASP sobre los resultados del algoritmo voraz) de 4 a 1: 4 tareas en promedio ejecutadas por recurso.

Los ***problemas test*** son pues, las instancias de prueba que se utilizarán en los experimentos y están conformados por la cantidad de máquinas, de tareas y por la matriz de tiempos de ejecución para los elementos involucrados. Los valores que se manejarán serán los siguientes:

- Número de tareas N: en el intervalo 100...250 tomando como puntos de referencia los valores de 100, 150, 200, 250.
- Número de recursos M: un máximo de 50 recursos diferentes, tomando como puntos

de referencia los valores de 12, 25, 37, 50.

- Matriz de tiempos de proceso T: se generará de forma aleatoria con valores entre 1 y 100 unidades de tiempo.¹⁶

En total tenemos 16 diferentes juegos de instancias para las combinaciones recursos-tareas. De la misma forma, para cada combinación se generarán 10 instancias distintas, lo que arroja un total de 160 problemas test realizados.

6.4. Calibración de las constantes de relajación α y θ .

El algoritmo GRASP maneja dos constantes α y θ , cuyo funcionamiento fue discutido en capítulos previos. El valor que tomen permitirá refinar la solución que genere la etapa de Construcción GRASP y facilitará su comparación con la solución voraz; de allí que no hemos determinado un único valor de dichas constantes para todas las instancias con las que hemos realizado los experimentos numéricos, sino más bien se han calibrado dichas constantes para cada caso en particular.

6.4.1. Proceso de calibración para una instancia determinada

Para establecer los mejores valores de α y θ se sucedieron los siguientes pasos:

- Seleccionar una instancia de trabajo: N, M y matriz T.
- Fijar el valor de α y θ en 0
- Ejecutar el algoritmo GRASP con dichos valores e incrementar el valor de θ en pasos iguales a 0.01 hasta llegar a 0.5. Los incrementos podrían variar. Consideramos como tope máximo 0,5 porque un mayor valor (sobre todo de α) aplicaría un elevado *carácter aleatorio* a la fase de construcción, lo que distorsionaría los resultados finales al ser enfrentados a los resultados del algoritmo voraz.
- Almacenar el valor arrojado por el algoritmo GRASP, el valor de α y el de θ .
- Repetir el proceso para el siguiente valor de α , el cual también sufrirá un incremento de 0.01

En total, se realizan **2500** experiencias cada vez que se quiera calibrar valores de α y de θ . Finalmente se escogen los menores makespan GRASP y se trabaja con los valores de α y θ utilizados. Así por ejemplo: para N = 100 tareas y M = 12 recursos se obtuvo entre otros, los resultados que se advierten en la siguiente tabla:

Tabla 6□1: Proceso de calibración para una instancia ejemplo 100x12

¹⁶ : Recordemos que un tiempo de ejecución muy alto ($+\infty$) puede interpretarse como que el recurso no puede ejecutar alguna tarea. Usar en este caso, un tiempo de ejecución igual a 0 podría confundirse como que recurso ejecuta *tan rápido la tarea* que se puede asumir que lo hace de forma instantánea, sin ocupar tiempo (N. del A.).

GRASP	α	θ
198	0.32	0.01
202	0.35	0
202	0.37	0
203	0.05	0.01
203	0.06	0.01
203	0.09	0.01
203	0.13	0.01
203	0.14	0
203	0.19	0
203	0.24	0
203	0.29	0.02
203	0.39	0.02
203	0.4	0.01

Desde los valores tabulados, se infiere que los mejores resultados de la aplicación del algoritmo GRASP construcción se produjeron para valores de $\alpha = 0.32$ y $\theta = 0.01$; por lo tanto podríamos usarlos para las pruebas con instancias 100x12 en promedio, sin perder generalidad. Se puede establecer igualmente, un promedio entre los dos o tres mejores valores de las constantes. A través del sistema se puede realizar este proceso:



Figura 6□1: Calibración por medio del sistema

Ingresando los datos requeridos el sistema forma los 2500 valores y escoge los mejores presentándolos a continuación. Se puede partir de un valor determinado de α y θ para acotar la calibración (por defecto se parte de valores iguales a 0)

Computacionalmente, el proceso de calibración ocupa un alto porcentaje de la memoria para instancias grandes. Cuando se calibraban las instancias de prueba mayores no se han podido realizar más de dos calibraciones seguidas, sin que el sistema operativo indicase una disminución crítica de la memoria virtual.



Figura 6□2: Ejemplo de mejores valores de α y θ calibrados para una instancia

Al presentar los resultados resúmenes, se mostrará los valores de α y θ empleados para los experimentos.

6.5. Resultados de los Experimentos

En la siguiente tabla resumen se muestra los resultados promedios de la ejecución de los algoritmos voraz y GRASP (en ese orden) sobre las instancias respectivas; se listan también, los valores usados de las constantes α y θ y finalmente la eficiencia promedio del resultado GRASP sobre el resultado voraz

$$\left(1 - \frac{\text{ResultadoGRASP}}{\text{ResultadoGobso}}\right)$$

Este resumen se ha obtenido, como mencionamos antes, de los promedios arrojados por las 10 instancias de cada uno de los 16 problemas test desarrollados:

Tabla 6□2: Resumen de resultados de los experimentos numéricos

Tareas/ Recursos	VORAZ	GRASP C			EFICIENCIA
	Makespan	Makespan	α	θ	%
100 \ 12	213.3	193.7	0.26	0.04	9.19%
100 \ 25	113.7	108.6	0.24	0.01	4.49%
100 \ 37	76.5	74.3	0.155	0.01	2.88%
100 \ 50	59.1	57.8	0.145	0.01	2.20%
150 \ 12	283.1	258.2	0.25	0.01	8.80%
150 \ 25	125.2	114	0.2055	0.01	8.95%
150 \ 37	86.1	84.1	0.155	0.01	2.32%
150 \ 50	68.6	67.4	0.115	0.01	1.75%
200 \ 12	325.9	307	0.07	0.01	5.80%
200 \ 25	127	117	0.247	0.01	7.87%
200 \ 37	109.8	105.2	0.112	0.01	4.19%
200 \ 50	76.4	74.6	0.155	0.01	2.36%
250 \ 12	411.8	377.1	0.15	0.01	8.43%
250 \ 25	183.5	168.2	0.28	0.01	8.34%
250 \ 37	125.3	119.6	0.26	0.01	4.55%
250 \ 50	96.5	91.1	0.123	0.01	5.60%
Eficiencia algoritmo GRASP sobre el algoritmo Voraz					5.48%

6.6. Calidad real de las soluciones

Ya hemos demostrado a partir de los experimentos, que el algoritmo GRASP supera en un promedio de 5 por ciento al rendimiento del algoritmo voraz. Sin embargo, mientras no tuviésemos alguna referencia de la solución exacta para algunas instancias no podríamos mencionar la real dimensión de la calidad de dicho algoritmo.

Es por eso que, ante la no existencia en la literatura de instancias de prueba para el problema en cuestión (sea para el caso de proyectos de software o para aplicaciones de tipo industrial) se decidió crear instancias lo suficientemente pequeñas (manejables) para probarlas como problemas de la programación lineal de la mano del modelo matemático presentado en capítulos anteriores para la variante que se ha desarrollado.

Así pues, se plantearon instancias que iban a ser resueltas por medio del programa **LINDO** [Scha97] para obtener sus soluciones exactas. Luego se compararía esas soluciones exactas¹⁷ con los resultados arrojados por nuestros algoritmos y así obtener la eficiencia real de los mismos.

¹⁷ : Hacemos la salvedad nuevamente, que el modelo matemático tiene un carácter de relajado por lo que no se podría considerar los resultados obtenidos en LINDO como *exactos al 100%*. Sin perder generalidad, sirven como medida de comparación para los algoritmos propuestos (N. del A.)

6.6.1. Instancias usadas. Rango de tamaños

Las instancias usadas para ser resueltas de forma exacta han tenido los siguientes tamaños (recuérdese que N es el número de tareas y M el número de máquinas):

N	M
6	3
8	3
12	3
15	3
10	5
15	5
20	5
25	5

Tabla 6□3: Tamaños de las instancias de prueba exacta

La limitante ha sido la misma capacidad del software de programación lineal LINDO en su versión 6.0 estudiantil para entorno MS Windows. Esto debido a que las matrices de tiempo de trabajo se convertían en restricciones del modelo. Por ejemplo: para una instancia de 25x5 teníamos una restricción muy grande con 126 términos (una inecuación) y luego 25 restricciones con 5 términos (restricciones de asignación) para luego introducir las restricciones de precedencia (0 ó mas por cada una de las 25 tareas).

6.6.2. Ejemplo de una instancia modelada en LINDO

Vamos a presentar en la siguiente figura, una instancia de prueba con los siguientes datos: N = 6, M = 3; matriz de tiempos de ejecución T:

	M1	M2	M3
1	27	3	95
2	46	67	65
3	99	41	51
4	18	29	56
5	11	18	77
6	93	82	16

Tabla 6□4: Una matriz de tiempos T para una instancia 6x3

Y una lista de predecesoras como sigue:

Predecesoras	
J1	No tiene
J2	J1 J4
J3	J5 J6
J4	J6
J5	J6
J6	No tiene

Tabla 6□5: Lista de predecesoras para una instancia 6x3

Su presentación siguiendo como un problema de programación lineal sería:

```

1. min X0
subject to
2. 27 X11 + 46 X12 + 89 X31 + 78 X41 + 77 X51 + 93 X61 - A1 = 0
   3 X12 + 47 X22 + 41 X32 + 29 X42 + 18 X52 + 82 X62 - A2 = 0
   95 X13 + 45 X23 + 51 X33 + 56 X43 + 71 X53 + 44 X63 - A3 = 0
   X0 - A1 = 0
   X0 - A2 = 0
   X0 - A3 = 0
3. x11 + x12 + x13 = 1
   x21 + x22 + x23 = 1
   x31 + x32 + x33 = 1
   x41 + x42 + x43 = 1
   x51 + x52 + x53 = 1
   x61 + x62 + x63 = 1
4. x21 + x22 + x23 - x31 - x32 - x33 <= 0
   x21 + x22 + x23 - x41 - x42 - x43 <= 0
   x31 + x32 + x33 - x51 - x52 - x53 <= 0
   x31 + x32 + x33 - x61 - x62 - x63 <= 0
   x41 + x42 + x43 - x61 - x62 - x63 <= 0
   x51 + x52 + x53 - x61 - x62 - x63 <= 0

```

Figura 6□3: Modelo de programación lineal para la instancia de ejemplo 6x3

Algunos comentarios sobre el modelo anterior:

- La línea 1 cubre el objetivo del modelo que es la minimización de la variable X0 que representa aquí al makespan.
- La línea 2 presenta cada entrada del vector de tiempos acumulados de ejecución A. Se cumple que para cada entrada: $A_i \geq \sum_j T_{ij}$, para $j \in 1..M, i \in 1..N$. Luego, siguiendo la definición cada una de estas entradas debe ser menor que el makespan X0.
- La línea 3 presenta las restricciones de asignación: solamente una de las entradas x_{ij} será igual a 1 mientras que el resto serán igual a 0. Precisamente será igual a 1 en la entrada que corresponda a la máquina j-ésima ejecutora de la tarea i-ésima.
- La línea 4 representa a las restricciones de precedencia.

Ahora mostraremos la solución obtenida a partir para el modelo, por el LINDO en su versión reducida:

```

1. L.P. OPTIMUM FOUND AT STEP 61
2. OBJECTIVE FUNCTION VALUE 67.00000
3. VARIABLE VALUE REDUCED COST
   X12 1.000000 -1.000000
   X31 1.000000 0.000000
   X33 1.000000 51.000000
   X42 1.000000 0.000000
   X52 1.000000 0.000000
   X63 1.000000 16.000000
   X0 67.000000 0.000000
   A1 46.000000 0.000000
   A2 58.000000 0.000000
   A3 67.000000 0.000000
NO. ITERATIONS= 61
BRANCHES= 5 DETERM= 1.0000 0

```

Figura 6□4: Resultado exacto para la instancia ejemplo 6x3

Adviértase lo siguiente:

- En la línea 1 se muestra la cantidad de iteraciones que se realizaron para encontrar la

solución actual.

- En la línea 2 se muestra el valor de la función objetivo. Para nuestro problema representará al makespan: el menor tiempo de trabajo acumulado de algún recurso.
- A partir de la línea 3 se muestra el valor que ha obtenido cada una de las variables X_{ij} del modelo: un valor igual a 1 significa que la máquina j ejecutó la tarea i ; un valor igual a 0 indicará el caso contrario ¹⁸.

6.6.3. Cuadro resumen de las experiencias realizadas con LINDO.

El siguiente cuadro resume la información de las instancias usadas para las pruebas buscando valores exactos. Mostramos el tamaño de la instancia (NxM) así como las soluciones arrojadas por el programa LINDO, el algoritmo voraz, el algoritmo GRASP y los porcentajes de eficiencia caso los hubiere. En primer lugar presentamos las instancias con M igual a 3

Matriz	n	m	Resultado del LINDO	Algoritmo voraz	Algoritmo GRASP	% Exactitud	% Exactitud GRASP	σ	δ
Inst. 1	10	3	10	10	10	100.0%	100.0%	0.00	0.00
Inst. 2	10	3	10	10	10	100.0%	100.0%	0.00	0.00
Inst. 3	10	3	10	10	10	100.0%	100.0%	0.00	0.00
Inst. 4	10	3	10	10	10	100.0%	100.0%	0.00	0.00
Inst. 5	10	3	10	10	10	100.0%	100.0%	0.00	0.00
Inst. 6	10	3	10	10	10	100.0%	100.0%	0.00	0.00
Inst. 7	10	3	10	10	10	100.0%	100.0%	0.00	0.00
Inst. 8	10	3	10	10	10	100.0%	100.0%	0.00	0.00
Inst. 9	10	3	10	10	10	100.0%	100.0%	0.00	0.00
Inst. 10	10	3	10	10	10	100.0%	100.0%	0.00	0.00
Inst. 11	10	3	10	10	10	100.0%	100.0%	0.00	0.00
Inst. 12	10	3	10	10	10	100.0%	100.0%	0.00	0.00
Inst. 13	10	3	10	10	10	100.0%	100.0%	0.00	0.00
Inst. 14	10	3	10	10	10	100.0%	100.0%	0.00	0.00
Inst. 15	10	3	10	10	10	100.0%	100.0%	0.00	0.00
Inst. 16	10	3	10	10	10	100.0%	100.0%	0.00	0.00
Inst. 17	10	3	10	10	10	100.0%	100.0%	0.00	0.00
Inst. 18	10	3	10	10	10	100.0%	100.0%	0.00	0.00
Inst. 19	10	3	10	10	10	100.0%	100.0%	0.00	0.00
Inst. 20	10	3	10	10	10	100.0%	100.0%	0.00	0.00
Inst. 21	10	3	10	10	10	100.0%	100.0%	0.00	0.00
Inst. 22	10	3	10	10	10	100.0%	100.0%	0.00	0.00
Inst. 23	10	3	10	10	10	100.0%	100.0%	0.00	0.00
Inst. 24	10	3	10	10	10	100.0%	100.0%	0.00	0.00
Inst. 25	10	3	10	10	10	100.0%	100.0%	0.00	0.00
Inst. 26	10	3	10	10	10	100.0%	100.0%	0.00	0.00
Inst. 27	10	3	10	10	10	100.0%	100.0%	0.00	0.00
Inst. 28	10	3	10	10	10	100.0%	100.0%	0.00	0.00
Inst. 29	10	3	10	10	10	100.0%	100.0%	0.00	0.00
Inst. 30	10	3	10	10	10	100.0%	100.0%	0.00	0.00
Inst. 31	10	3	10	10	10	100.0%	100.0%	0.00	0.00
Inst. 32	10	3	10	10	10	100.0%	100.0%	0.00	0.00
Inst. 33	10	3	10	10	10	100.0%	100.0%	0.00	0.00
Inst. 34	10	3	10	10	10	100.0%	100.0%	0.00	0.00
Inst. 35	10	3	10	10	10	100.0%	100.0%	0.00	0.00
Inst. 36	10	3	10	10	10	100.0%	100.0%	0.00	0.00
Inst. 37	10	3	10	10	10	100.0%	100.0%	0.00	0.00
Inst. 38	10	3	10	10	10	100.0%	100.0%	0.00	0.00
Inst. 39	10	3	10	10	10	100.0%	100.0%	0.00	0.00
Inst. 40	10	3	10	10	10	100.0%	100.0%	0.00	0.00
Inst. 41	10	3	10	10	10	100.0%	100.0%	0.00	0.00
Inst. 42	10	3	10	10	10	100.0%	100.0%	0.00	0.00
Inst. 43	10	3	10	10	10	100.0%	100.0%	0.00	0.00
Inst. 44	10	3	10	10	10	100.0%	100.0%	0.00	0.00
Inst. 45	10	3	10	10	10	100.0%	100.0%	0.00	0.00
Inst. 46	10	3	10	10	10	100.0%	100.0%	0.00	0.00
Inst. 47	10	3	10	10	10	100.0%	100.0%	0.00	0.00
Inst. 48	10	3	10	10	10	100.0%	100.0%	0.00	0.00
Inst. 49	10	3	10	10	10	100.0%	100.0%	0.00	0.00
Inst. 50	10	3	10	10	10	100.0%	100.0%	0.00	0.00
Inst. 51	10	3	10	10	10	100.0%	100.0%	0.00	0.00
Inst. 52	10	3	10	10	10	100.0%	100.0%	0.00	0.00
Inst. 53	10	3	10	10	10	100.0%	100.0%	0.00	0.00
Inst. 54	10	3	10	10	10	100.0%	100.0%	0.00	0.00
Inst. 55	10	3	10	10	10	100.0%	100.0%	0.00	0.00
Inst. 56	10	3	10	10	10	100.0%	100.0%	0.00	0.00
Inst. 57	10	3	10	10	10	100.0%	100.0%	0.00	0.00
Inst. 58	10	3	10	10	10	100.0%	100.0%	0.00	0.00
Inst. 59	10	3	10	10	10	100.0%	100.0%	0.00	0.00
Inst. 60	10	3	10	10	10	100.0%	100.0%	0.00	0.00
Inst. 61	10	3	10	10	10	100.0%	100.0%	0.00	0.00
Inst. 62	10	3	10	10	10	100.0%	100.0%	0.00	0.00
Inst. 63	10	3	10	10	10	100.0%	100.0%	0.00	0.00
Inst. 64	10	3	10	10	10	100.0%	100.0%	0.00	0.00
Inst. 65	10	3	10	10	10	100.0%	100.0%	0.00	0.00
Inst. 66	10	3	10	10	10	100.0%	100.0%	0.00	0.00
Inst. 67	10	3	10	10	10	100.0%	100.0%	0.00	0.00
Inst. 68	10	3	10	10	10	100.0%	100.0%	0.00	0.00
Inst. 69	10	3	10	10	10	100.0%	100.0%	0.00	0.00
Inst. 70	10	3	10	10	10	100.0%	100.0%	0.00	0.00
Inst. 71	10	3	10	10	10	100.0%	100.0%	0.00	0.00
Inst. 72	10	3	10	10	10	100.0%	100.0%	0.00	0.00
Inst. 73	10	3	10	10	10	100.0%	100.0%	0.00	0.00
Inst. 74	10	3	10	10	10	100.0%	100.0%	0.00	0.00
Inst. 75	10	3	10	10	10	100.0%	100.0%	0.00	0.00
Inst. 76	10	3	10	10	10	100.0%	100.0%	0.00	0.00
Inst. 77	10	3	10	10	10	100.0%	100.0%	0.00	0.00
Inst. 78	10	3	10	10	10	100.0%	100.0%	0.00	0.00
Inst. 79	10	3	10	10	10	100.0%	100.0%	0.00	0.00
Inst. 80	10	3	10	10	10	100.0%	100.0%	0.00	0.00
Inst. 81	10	3	10	10	10	100.0%	100.0%	0.00	0.00
Inst. 82	10	3	10	10	10	100.0%	100.0%	0.00	0.00
Inst. 83	10	3	10	10	10	100.0%	100.0%	0.00	0.00
Inst. 84	10	3	10	10	10	100.0%	100.0%	0.00	0.00
Inst. 85	10	3	10	10	10	100.0%	100.0%	0.00	0.00
Inst. 86	10	3	10	10	10	100.0%	100.0%	0.00	0.00
Inst. 87	10	3	10	10	10	100.0%	100.0%	0.00	0.00
Inst. 88	10	3	10	10	10	100.0%	100.0%	0.00	0.00
Inst. 89	10	3	10	10	10	100.0%	100.0%	0.00	0.00
Inst. 90	10	3	10	10	10	100.0%	100.0%	0.00	0.00
Inst. 91	10	3	10	10	10	100.0%	100.0%	0.00	0.00
Inst. 92	10	3	10	10	10	100.0%	100.0%	0.00	0.00
Inst. 93	10	3	10	10	10	100.0%	100.0%	0.00	0.00
Inst. 94	10	3	10	10	10	100.0%	100.0%	0.00	0.00
Inst. 95	10	3	10	10	10	100.0%	100.0%	0.00	0.00
Inst. 96	10	3	10	10	10	100.0%	100.0%	0.00	0.00
Inst. 97	10	3	10	10	10	100.0%	100.0%	0.00	0.00
Inst. 98	10	3	10	10	10	100.0%	100.0%	0.00	0.00
Inst. 99	10	3	10	10	10	100.0%	100.0%	0.00	0.00
Inst. 100	10	3	10	10	10	100.0%	100.0%	0.00	0.00
Inst. 101	10	3	10	10	10	100.0%	100.0%	0.00	0.00
Inst. 102	10	3	10	10	10	100.0%	100.0%	0.00	0.00
Inst. 103	10	3	10	10	10	100.0%	100.0%	0.00	0.00
Inst. 104	10	3	10	10	10	100.0%	100.0%	0.00	0.00
Inst. 105	10	3	10	10	10	100.0%	100.0%	0.00	0.00
Inst. 106	10	3	10	10	10	100.0%	100.0%	0.00	0.00
Inst. 107	10	3	10	10	10	100.0%	100.0%	0.00	0.00
Inst. 108	10	3	10	10	10	100.0%	100.0%	0.00	0.00
Inst. 109	10	3	10	10	10	100.0%	100.0%	0.00	0.00
Inst. 110	10	3	10	10	10	100.0%	100.0%	0.00	0.00
Inst. 111	10	3	10	10	10	100.0%	100.0%	0.00	0.00
Inst. 112	10	3	10	10	10	100.0%	100.0%	0.00	0.00
Inst. 113	10	3	10	10	10	100.0%	100.0%	0.00	0.00
Inst. 114	10	3	10	10	10	100.0%	100.0%	0.00	0.00
Inst. 115	10	3	10	10	10	100.0%	100.0%	0.00	0.00
Inst. 116	10	3	10	10	10	100.0%	100.0%	0.00	0.00
Inst. 117	10	3	10	10	10	100.0%	100.0%	0.00	0.00
Inst. 118	10	3	10	10	10	100.0%	100.0%	0.00	0.00
Inst. 119	10	3	10	10	10	100.0%	100.0%	0.00	0.00
Inst. 120	10	3	10	10	10	100.0%	100.0%	0.00	0.00
Inst. 121	10	3	10	10	10	100.0%	100.0%	0.00	0.00
Inst. 122	10	3	10	10	10	100.0%	100.0%	0.00	0.00
Inst. 123	10	3	10	10	10	100.0%	100.0%	0.00	0.00
Inst. 124	10	3	10	10	10	100.0%	100.0%	0.00	0.00
Inst. 125	10	3	10	10	10	100.0%	100.0%	0.00	0.00
Inst. 126	10	3	10	10	10	100.0%	100.0%	0.00	0.00
Inst. 127	10	3	10	10	10	100.0%	100.0%	0.00	0.00
Inst. 128	10	3	10	10	10	100.0%	100.0%	0.00	0.00
Inst. 129	10	3	10	10	10	100.0%	100.0%	0.00	0.00
Inst. 130	10	3	10	10	10	100.0%	100.0%	0.00	0.00
Inst. 131	10	3	10	10	10	100.0%	100.0%	0.00	0.00
Inst. 132	10	3	10	10	10	100.0%	100.0%	0.00	0.00
Inst. 133	10	3	10	10	10	100.0%	100.0%	0.00	0.00
Inst. 134	10	3	10	10	10	100.0%	100.0%	0.00	0.00
Inst. 135	10	3	10	10	10	100.0%	100.0%	0.00	0.00
Inst. 136	10	3	10	10	10	100.0%	100.0%	0.00	0.00
Inst. 137	10	3	10	10	10	100.0%	100.0%	0.00	0.00
Inst. 138	10	3	10	10	10	100.0%	100.0%	0.00	0.00
Inst. 139	10	3	10	10	10	100.0%	100.0%	0.00	0.00
Inst. 140	10	3	10	10	10	100.0%	100.0%	0.00	0.00
Inst. 141	10	3	10	10	10	100.0%	100.0%	0.00	0.00
Inst. 142	10	3	10	10	10	100.0%	100.0%	0.00	0.00
Inst. 143	10	3	10	10	10				

Capítulo VII. Análisis de los experimentos realizados

7.1. Análisis de la eficiencia de los algoritmos entre sí

En el 100% los casos, la aplicación del algoritmo GRASP supera al comportamiento del algoritmo voraz. Se desprenden los siguientes postulados:

- Los resultados obtenidos tras la aplicación del algoritmo voraz son buenos, situación que puede ser comprobada a través del empleo de software especializado en programación entera (Lindo, Super Lindo) en donde las soluciones se acercan a las soluciones exactas. No obstante, esta comparación no se puede aplicar para las instancias empleadas en la tesis debido a su tamaño (1200 y 12500 variables por instancia). Asumimos que era necesario efectuar pruebas con instancias más pequeñas que sí pueden ser resueltas desde el modelo matemático planteado en capítulos anteriores.
- El promedio total de eficiencia alcanzado por el algoritmo GRASP es de 5.48 % sobre los resultados de aplicación del algoritmo voraz, que encaja dentro de los cálculos hechos durante el proyecto, en base a los resultados obtenidos en anteriores

investigaciones del autor [Tu01].

- La eficiencia del algoritmo GRASP disminuye a medida que aumenta el número de recursos, para un mismo número de tareas. Veamos por ejemplo, el comportamiento presentado en el siguiente gráfico para el caso de N igual a 200 tareas:

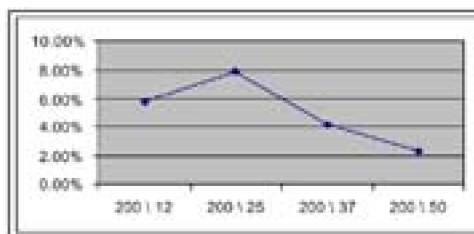


Figura 7-1: Gráfico de la eficiencia N = 200

El tiempo de CPU es muy bajo para el caso de la aplicación del algoritmo voraz pues apenas llega a medio segundo. El tiempo de CPU para el algoritmo GRASP llega a un máximo de 5 minutos, y en donde la fase de construcción consume la mayoría de él. Debemos recordar que esta fase se ejecuta un máximo de 2500 veces, tanto para la aplicación directa como para el proceso de calibración lo que arrojaría en promedio 0.12 segundos por iteración, tiempo igualmente bajo.

Para un mayor detalle de los experimentos, véase el **anexo A**.

7.1.1. Criterios de relajamiento empleados

Como parte de los nuevos planteamientos presentados en el algoritmo GRASP se tiene:

- Un criterio de selección de la mejor tarea a programar (usando constante $\square$)
- Un criterio de selección del mejor recurso que puede ejecutar la tarea seleccionada con el criterio anterior (usando nueva constante θ)

En la literatura no existen algoritmos GRASP que consideren un doble criterio de relajamiento al momento de hacer las selecciones correspondientes: las aplicaciones convencionales de GRASP, solamente forman una lista RCL de candidatas para las tareas a ser ejecutadas (para otras variantes del problema del scheduling mismo).

El presente trabajo de tesis amplía esa visión al proporcionarle un componente de adaptabilidad y aleatorio a la selección de la mejor máquina ejecutora; lo que al final afectó positivamente a la eficiencia del algoritmo.

7.1.2. Rendimiento de los algoritmos

El algoritmo voraz arroja soluciones que están a un promedio de entre 18% y 28% de la solución exacta para instancias de bajo tamaño. El tiempo que le ocupa al computador ejecutar este algoritmo es insignificante para casi cualquier instancia de prueba (por debajo de los 2 segundos).

Para el algoritmo voraz, el escoger primero las tareas de mayor duración para evitar perturbaciones al final ¹⁹, favorece a los resultados **únicamente en los casos de instancias relativamente pequeñas (como máximo 50 tareas)** y en donde la proporción de predecesoras por tarea es alta (en promedio 8). En el resto de casos, como las instancias que fueron sujetas a experimentos, empeora el resultado voraz: es por eso que se prefirió omitir la selección de las tareas más pesadas para tener los mejores resultados posibles de dicho algoritmo.

Los resultados de la aplicación del algoritmo GRASP mejoran en un 100% a los resultados del algoritmo voraz para instancias de prueba suficientemente elevadas (proporción 4 a 1, 5 a 1). En instancias más pequeñas como mínimo igualan en resultados, es superado en un bajísimo porcentaje. El porcentaje de superioridad del algoritmo GRASP frente al algoritmo voraz es de 5% en promedio.

El algoritmo GRASP se acerca mucho más a las soluciones exactas para las instancias analizadas, quedando en el rango promedio entre 5% y 9% de dichas soluciones. En múltiples ocasiones iguala a la solución obtenida a través de la ejecución del modelo.

El tiempo de ejecución de la fase de construcción GRASP se basa en la cantidad de iteraciones a realizarse: para las instancias mayores (250 x 50) se trataron de realizar más de 1000 iteraciones, **pero aparentemente la configuración del hardware de prueba no lo permitió**. Esto nos lleva a inferir que, con una configuración de equipo más potente sí se podría pasar esta barrera: el tiempo de CPU empleado no pasaba de los 3 minutos lo que da un promedio bajo para esta fase del algoritmo.

La etapa de mejoría GRASP, se limitó a un intercambio de los valores encontrados en la etapa anterior de construcción. **No brindó mejoras en ninguno de los casos**, lo que nos conduce a pensar que la etapa de construcción es lo suficientemente robusta, para requerir algún tipo de permutaciones que busquen mejorar su resultado. Es por eso que no se utilizó la mejora.

7.2. Análisis de eficiencia real

Continuado con el análisis de las soluciones obtenidas por la aplicación del algoritmo GRASP propuesto, ahora se hará en función de dos parámetros: la cantidad de iteraciones que se ejecuta la fase de construcción y el valor de la constante α empleados.

7.2.1. Calidad de la solución versus el número de iteraciones de la fase de construcción GRASP

Nuestro interés por saber cómo variaba la eficiencia de la solución GRASP respecto a la

¹⁹ : Técnica empleada en el planteamiento de la GRASP desarrollada por el autor, para la variante de tareas independientes y máquinas heterogéneas [Tu01].

cantidad de iteraciones realizadas nos llevaron a realizar varias corridas del algoritmo con diferentes valores para la cantidad de repeticiones hechas. El procedimiento seguido fue el siguiente:

- Se selecciona una matriz de cada instancia de las presentadas originalmente
- Se efectúan sucesivas ejecuciones del algoritmo GRASP para los mejores valores calibrados de α y θ respectivamente.
- Las iteraciones realizadas partían desde 1 hasta *el tope máximo que permitiese el hardware donde se ejecutaron las pruebas para el tamaño de la instancia*. Esta limitante ya se comentó en apartados anteriores. Los incrementos en las iteraciones se hacía de 250 o 500 por paso.

En la siguiente figura vemos una gráfica de ejemplo con la evolución del porcentaje de eficiencia para una instancia de tamaño 100x12 (para más detalles ver el [Anexo B](#))

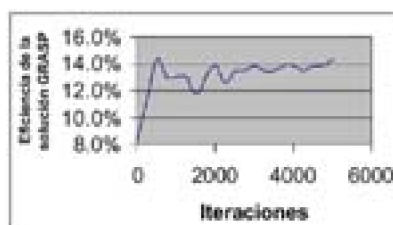


Figura 7-2: Eficiencia versus iteraciones GRASP para una instancia de 100x12

El comportamiento ilustrado nos indica variaciones muy oscilantes hasta aproximarse a las 5000 iteraciones donde parece que el comportamiento de la eficiencia se estabiliza rondando el 15% de eficiencia. En las conclusiones analizaremos los efectos de las iteraciones con respecto también, al tamaño de la instancia.

7.2.2. Calidad de la solución versus los valores de la constante de relajación α

A partir del resultado anterior se determina la cantidad óptima de iteraciones para cada una de las instancias de prueba. Con ese valor se procedió a realizar un número finito de pruebas variando el valor de la constante α . Aquí se describe el procedimiento:

- Se selecciona una matriz de cada instancia de las presentadas originalmente
- Hacemos sucesivas ejecuciones del algoritmo GRASP para *los mejores valores de las iteraciones de la fase de construcción* calculados como se menciona en el apartado anterior.
- Se parte de un valor de α igual a 0 y se lo va variando con pasos iguales a 0.01 unidades entre valor y valor. En la siguiente gráfica se ve el comportamiento de la eficiencia para una instancia ejemplo de orden 100x12:

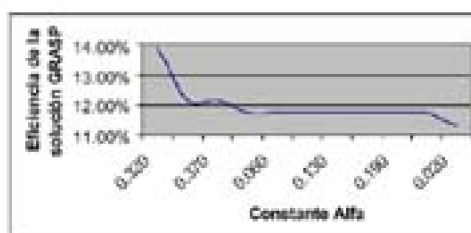


Figura 7-3: Eficiencia versus valor de α para una instancia de 100x12

Como puede observarse, es muy corto el intervalo de valores de α en donde se logra la mayor eficiencia, comportamiento que se encontró en todas las experiencias, siendo entonces vital la elección de valores adecuados de ambas constantes de relajación (ver [Anexo B](#) para mayores detalles)

7.3. Análisis de los resultados experimentales desde la perspectiva de la Ingeniería de Software

Se pueden deslizar algunos comentarios sobre el rendimiento de los algoritmos desde el punto de vista de la administración de proyectos de desarrollo de software:

- La cantidad de tareas involucradas en un proyecto de desarrollo no es baja. Por lo tanto, no se podría aplicar ningún método de programación no lineal para tentar obtener soluciones exactas.
- La cantidad de tareas asignadas comúnmente a cada recursos (humano) dentro de un proyecto tampoco es tan baja²⁰, otra razón para la aplicación del algoritmo GRASP propuesto.

En proyectos de consultoría muy grandes, la cantidad de recursos involucrados puede ser muy elevada lo que implicaría una administración *horizontal* muy precisa para el control de las tareas y los artefactos (planillas, formatos) que se espera como producción de cada uno de los mencionados recursos.

²⁰ : Se puede pensar que por ejemplo, a un programador se le asigna **una única tarea: programar**, cuando en realidad se le están asignando varios módulos (N. del A.)

Capítulo VIII Conclusiones

8.1. Conclusiones sobre el proyecto

En este capítulo se presentan las conclusiones finales obtenidas a partir de las experiencias numéricas y el rendimiento de los algoritmos propuestos. De igual forma, se plantean algunas posibles aplicaciones en la Industria de Software en el Perú, en base a la implementación de sistemas conteniendo dichos algoritmos; se encauzan finalmente, temas de futuras investigaciones posibles.

Los sistemas existentes en el mercado que pueden ser considerados como planificadores de tareas, no han aplicado técnicas meta heurísticas sino más bien métodos exactos. No buscan la optimización del ordenamiento de las tareas a ejecutar, sino más bien dan énfasis a encontrar planes factibles de ser ejecutados, analizando la carga de los recursos involucrados (MS PROJECT, PRIMAVERA PROJECT, etc.); sean estas planificaciones para proyectos de Ingeniería de Software y líneas de producción industrial.

La mayoría de las consultoras de sistemas emplean los productos que ya existen en el mercado y se ajustan a las particularidades de cada uno de ellos para planificar cada una de las etapas de sus proyectos, sin importar la metodología a la que recurran.

Este panorama le imprime un carácter novedoso a la investigación realizada ya que,

si bien en la actualidad las técnicas GRASP son frecuentemente empleadas para la resolución de distintas clases de problemas, hasta ahora no se había planteado una GRASP para resolver el problema del task scheduling para tareas involucradas en proyectos de desarrollo de sistemas.

No existen, similarmente, sistemas que planifiquen una línea de producción completa considerando entre otros el desplazamiento de productos-insumos de la línea, la ejecución de las operaciones, la disposición de la maquinaria (layout) y de las facilidades (instalaciones), penalidades y limitaciones en la carga de trabajo de las máquinas, reducción de tiempos muertos, etc.

8.2. Conclusiones sobre los algoritmos propuestos

En la tesis se ha propuesto dos algoritmos para el problema de la planificación de los recursos en proyectos de Ingeniería de Software (variación del task scheduling):

- Un algoritmo voraz
- Un algoritmo GRASP construcción, con doble criterio de relajación.

En la literatura no se encuentran publicados ninguno tipo similar de algoritmos, lo que constituye la principal contribución de la tesis. Sobre las características particulares de los algoritmos implementados se mencionan algunos conceptos a continuación.

8.3. Aplicabilidad en la Industria Peruana

Se ha venido mencionando que para un adecuado uso de los algoritmos presentados en este trabajo es necesario estudiar el entorno del problema que se va a tratar de resolver, definiendo adecuadamente las entidades que harán las veces de recursos ejecutores y las tareas. Visto esto, podemos apreciar el amplio espectro de industrias en donde se pueden aplicar los algoritmos propuestos.

En el tema de la Ingeniería de Software, el tema de la planificación de las tareas comprendidas en los proyectos de desarrollo es fundamental para garantizar la calidad y el éxito de los productos que se obtengan en el futuro. No es extraño que una mala planificación, como normalmente sucede en las consultoras medianas y pequeñas de nuestro país, encarezca los costos del proyecto, alargue innecesariamente los plazos de entrega en el tiempo y produzca un malestar a los profesionales que sufren sobrecarga de trabajo.

En otro tipo de industria, tales como:

- Industrias textiles.

- Industrias del empaque, envasado, embotellado.
- Industrias mineras y metal-mecánicas.
- En general cualquier línea de producción, ensamblaje, etc.

8.4. Futuras investigaciones: task scheduling y optimización en general

Queda pendiente sobre el tema de la programación de tareas, relacionarla de forma conveniente con los temas de layout, disposición de maquinaria y planificación de facilidades, y otros problemas similares que se complementan. Refiriéndonos exclusivamente al tema del task scheduling se deberían contemplar algoritmos en los que se analicen:

- Minimización de tiempos ociosos y muertos en las máquinas a la vez.
- Consideración de tiempos muertos o tiempos de recarga de determinadas máquinas, que la literatura denomina **penalizaciones**, y que ha sido considerado por ejemplo, en algunas GRASP presentadas para la variante del job scheduling.
- Un mayor nivel de restricciones: ciertas tareas tienen que pasar por determinadas máquinas; ciertas máquinas tienen un límite de tiempo de trabajo, etc.

8.5. Recomendaciones Finales

Uno de los motivos que impulsaron la realización de esta tesis fue el mostrar la potencialidad de la investigación en el campo de la Inteligencia Artificial y su aplicación directa en la industria de nuestro país, a costos relativamente moderados. La Universidad Nacional de San Marcos ha demostrado su alta capacidad investigación científica de en diversas ramas de las ciencias y sería beneficioso desarrollar una línea de investigación en este campo, tal que brinde servicios de asesoría industrial a terceros como hacen algunas otras universidades en otros campos con pequeñas empresas que asesoren directamente a las industrias y apliquen técnicas de IA con el fin de optimizar sus procesos.

Los temas presentados ya han sido desarrollados si bien no en su totalidad, en otros países de la región, como es el caso de Brasil, donde existen sistemas de programación de tareas para plantas de ensamblaje de autos y otras aplicaciones más de distintos problemas. La Universidad debería incentivar dado su carácter multidisciplinario. Véase:

- Para las etapas de análisis y programación se requerirá del concurso de ingenieros de sistemas e informática; expertos en Investigación de operaciones.

- Para la determinación de las tareas se requerirá de los conocimientos de ingenieros industriales.
- Para la determinación de las máquinas, su funcionamiento, capacidad y tiempos de proceso, así como para la delimitación de las células de trabajo operario – maquinaria será necesario el apoyo de ingenieros mecánicos, electrónicos, etc.
- Para los sistemas de identificación de patrones, de materiales o minerales así como de compuestos químicos se necesita de la intervención de ingenieros químicos.
- Para el desarrollo de sistemas expertos más elaborados, en donde los algoritmos creados serán parte de autómatas, es necesario el concurso de ingenieros electrónicos.

Como puede apreciarse los proyectos que se pueden emprender sobre estos temas pueden agrupar a los profesionales de todas las áreas de ingeniería de la Universidad, lo que ayudaría no sólo al país sino también a mantenernos a la vanguardia de la investigación científica. Esperamos haber contribuido para el inicio de esta loable labor.

BIBLIOGRAFÍA

- [Ale98] Allen P.; Stuart F. Component-Based Development for Enterprise Systems: Applying the Select Perspective. Cambridge University Press, USA, 1998.
- [Ba81] Ball M. The design and analysis of heuristics. Revista Networks, No. 11, feb-1981.
- [Bec00] Beck K. Planning eXtreme programming. First Edition. Addison-Wesley, USA, 2000.
- [BF89] Bard J., Feo T. Operations sequencing in discrete parts manufacturing. Journal of Management Science, 35: 249-255, 1989.
- [BF95] Bard J., Feo T., Holland S. Facility-wide planning and scheduling of printed wiring board assembly. Journal of Operations Research, 43(2): 219-230, 1995.
- [BGKRS95] Barr R., Golden B., Kelly J., Resende M., Stewart W. Designing and reporting on computational experiments with heuristic methods. Department of Computer Science and Engineering of Southern Methodist University of Dallas, USA, 1995.
- [Boo00] Booch G. Unifying Enterprise Development Teams with UML. Rational Software White Paper TP- 188 6/00. Disponible en <http://www.rational.com/worldwide>, 2000.
- [Boo90] Booch G. Object oriented Design with applications. Benjamin-Cumming Publishing Company, USA, 1990.

- [BRI00] Booch G., Rumbaugh J., Jacobson I. El proceso unificado de desarrollo de software. Addison-Wesley, USA, 1999.
- [BRI99] Booch G., Rumbaugh J., Jacobson I. El lenguaje unificado de modelado: manual de referencia. Addison-Wesley, USA, 1999.
- [CAIA90] Shah M. Knowledge-based dynamic scheduling in a steel plant. Proceeding of Conference on Artificial Intelligence Applications CAIA (6, 1990, USA) Editores Damian R., Silverman J, California-USA, p. 108-113, 1990.
- [Can01] Cantor M. RUP SE: The Rational Unified Process for Systems Engineering. Technical Report of Rational Community (on-line). Disponible en http://www.therationaledge.com/content/dec_01/m_RUPSEPart2_mc.html, 2001.
- [Can01b] Cantor M. The Rational Unified Process for Systems Engineering. PART II: Distinctive Features. Report of Rational Community (on-line). Disponible en http://www.therationaledge.com/content/dec_01/m_RUPSEPart2_mc.html, 2001.
- [Can03] Cantor M. Rational Unified Process for Systems Engineering. Part 1: Introducing RUP SE version 2.0. Technical Report of Rational Community (on-line). Disponible en http://www.therationaledge.com/content/aug_03/f_rupse_mc.jsp , 2003.
- [Can03b] Cantor M. Rational Unified Process for Systems Engineering. Part 2: System architecture. Technical Report of Rational Community (on-line). Disponible en http://www.therationaledge.com/content/sep_03//m_systemarch_mc.jsp , 2003.
- [Can03c] Cantor M. Rational Unified Process for Systems Engineering. Part 3: Requirements analysis and design. Technical Report of Rational Community (on-line). Disponible en http://www.therationaledge.com/content/oct_03/m_rupse_mc.jsp, 2003.
- [Cha03] Chan R. Adopting RUP in a COTS implementation project. Part 1: Getting started. Technical Report of Rational Community (on-line). Disponible en http://bronze.rational.com:8169/may_03/f_chan_new.jsp, 2003.
- [Char04] Charbonneau S. Software Project Management - A Mapping between RUP and the PMBOK. IBM-Rational White Paper. Disponible en <http://www-106.ibm.com/developerworks/rational/library/4721.html>, 2004.
- [CLEI04] Tupia M., Mauricio D. Un algoritmo GRASP para resolver el problema de la programación de tareas dependientes en máquinas diferentes. En las memorias de la Conferencia Latino Americana de Informática CLEI (30, 2004, Perú). Editores Solar M., Fernández-Baca D., Cuadros-Vargas E, Arequipa-Perú, p.129-139, 2004.
- [CLR01] Cormen T., Leiserson Ch., Rivest R. Introduction to Algorithms (2da edición). MIT Press - McGraw Hill, Inglaterra, 2001.
- [CM93] Campello R., Maculan N. Algoritmos e Heurísticas: desenvolvimento e avaliação de performance. Apolo Nacional, Brasil, 1993
- [Co67] Conway R. Theory of Scheduling. Addison-Wesley Publishing Company, USA, 1967.
- [Co02] Cockburn A. Agile software development. Addison-Wesley, USA, 2002.
- [DeM82] DeMarco T. Controlling Software Projects. Yourdon Press, USA, 1982.
- [Dav93] Davis A. Software Requirements: Objects, functions and states. Englewood Cliffs-Prentice Hall, USA, 1993.

-
- [ECP97] Koehler J., Nebel B, Hoffmann J. Extending planning graphs to an ADL subset
Proceeding of European Conference on Planning ECP (4, 1997, Francia) Editores
Nebel B., Hoffmann J., Dimopoulos Y, Toulouse- Francia, 1:273-285, 1997.
- [Eel04] Eeles P. Capturing Architectural Requirements. IBM-Rational White Paper.
Disponible en <http://www-128.ibm.com/developerworks/rational/library/4706.html>,
USA, 2004.
- [EP00] Eriksson H.; Penker M. Business Modeling with UML: Business Patterns at
Work. John Wiley & Son, USA, 2000.
- [FN71] Fikes R., Nilsson N. STRIPS a new approach to the application of theorem
providing to problem solving. Journal of Artificial Intelligence, 2(2): 3-4, 1971.
- [FR95] Feo T., Resende M.. Greedy Randomized Adaptive Search Procedure. Journal
of Global Optimization, 6:109-133, 1995.
- [FV91] Feo T., Venkatraman K. A Grasp for a difficult single machine scheduling
problem. Journal of Computer and Operation Research, No.18, 1991
- [Gau89] Gause D. Exploring requirements: Quality before design. Dorset House, USA,
1989.
- [GKL95] Glover F., Kelly J., Laguna M. Genetic Algorithms and taboo search: hybrids
for optimization. Journal of Computers and Operations Research, 1(22):111, 1995.
- [Hig03] Highsmith J. Adaptive Software Development: A Collaborative Approach to
Managing Complex Systems. Dorset House, 2000
- [HPL03] Hsieh J.C., Pei-Chann Ch., Lich-Chao H. Scheduling of drilling operations in
printed circuit board factory. Computers & Industrial Engineering, Pergamón, No. 44:
461–473, 2003.
- [Hum00] Humphrey W.S. Introduction to the Team Software Process. Addison-Wesley,
2000.
- [ICIL05] Tupia M. Un algoritmo voraz para resolver el problema de la programación de
tareas dependientes en máquinas diferentes. En las memorias de International
Conference on Industrial Logistics ICIL (7, 2005, Uruguay), (Poster abstract) Editor
Cancela H. Montevideo-Uruguay, p. 345, 2005.
- [ICM96] O'Connell W. Prospector: A content-based multimedia server for massively
parallel architectures. Proceeding from memories of ACM SIGMOD International
Conference on Management of Data Memories ICM (1996, Canadá) Editor Jeong T.,
Montreal – Canadá, p. 68-78, 1996
- [ICRA99] Pollard N.S. IEEE Grasp Parallel Algorithms for Synthesis of Whole-Hand
Grasps. Proceeding of International Conference on Robotics and Automation
ICRA(1999, USA), Albuquerque-USA. Disponible
<http://www.cs.brown.edu/people/~nsp/grasp.html>, 1999.
- [IEEE03] IEEE (Institute of Electric and Electronic Engineering) Standard Collection:
Software Engineering. Disponible <http://standards.ieee.org/software>, 2003.
- [IJCAI95] Blum A. Fast Planning through plan-graph analysis. Proceeding from
memories of International Joint Conference on Artificial Intelligence (14, 1995, USA),
editors Furst M, Blum A. Morgan Kaufmann, California – USA, p. 1636-1642, 1995.
- [Ja03] Jalali M, Afshar A. Reservoir Operation by Ant Colony Optimization Algorithms.

- Technical Report University of Science and Technology of Tehran, Iran, 2003.
- [JEJ95] Jacobson I., Ericsson M., Jacobson A. The Object Advantage: business process reengineering with object technology. Addison-Wesley, USA, 1995.
- [Joh04] Johnston S. Rational UML Profile for business modeling. IBM White Paper. Disponible en: <http://www-128.ibm.com/developerworks/rational/library/5167.html>, 2004.
- [Ka92] Kautz H., Selman B. Planning as Satisfiability. Technical Report N° AAA1-96 AT&T Labs Research. Disponible <http://www.research.att.com/~kautz/papers-ftp/satplan.ps>, 1992.
- [Kir66] Kirkpatrick T.; Clark N. PERT as aid for logic design. IBM Journal, USA, p.135-141, mar-1966.
- [Kro04] Kroll P. Planning and estimating a RUP project using IBM Rational SUMMIT Ascendant. IBM-Rational White Paper. Disponible en <http://www-128.ibm.com/developerworks/rational/library/4772.html>, 2004.
- [Kru00] Krutchen P. The Rational Unified Process. An Introduction. Addison-Wesley, USA, 2000.
- [Kru02] Krutchen P. Plannign an interative proyect. The Rational Edge. Technical Report of Rational Community (on-line). Disponible en http://www.therationaledge.com/content/oct_02/f_iterativePlanning_pk.jsp, USA, 2002.
- [Ku88] Kumara P. Artificial Intelligence: Manufacturing theory and practice. NorthCross Institute of Industrial Engineers, USA, p. 686, 1988
- [La91] Laguna M., Gonzalez J. A search heuristic for just-in-time scheduling in parallel machines. Journal of Intelligent Manufacturing, No. 2:253-260, 1991
- [La99] Larman C. UML y patrones: introducción al análisis y diseño orientado a objetos. Prentice Hall Hispanoamericana, México, 1999.
- [Lo99] Long D., M. Fox. The efficient implementation of the plan-graph in STAN. Journal of Artificial Intelligence, 10(1), 1999.
- [Lo03] Lopes E. Project planning best practices. The Rational Edge. Technical Report of Rational Community (on-line). Disponible en http://www.therationaledge.com/content/aug_03/m_projectplanning_dd_ec.jsp. USA, 2003.
- [McI00] McIntosh M. Content Management Using the Rational Unified Process. Rational Software White Paper TP- 164 9/00. Disponible en <http://www.rational.com/worldwide>, 2000.
- [MG60] Miller G., Galanter E.. Plans and the Structure of Behavior, Editorial Holt, New York-USA, 1960.
- [MIC01] Doerner K., Gutjahr W Ant Colony Optimization in Multiobjective Portfolio Selection, Proceeding of International Conference of Metaheuristics MIC (4, 2001, Portugal) p. 243-248, 2001.
- [MK91] Maux R., Keyes J. Handbook of Expert System in Manufacturing. McGraw-Hill. Madrid-España, p. 561, 1991.

-
- [Mom87] Mompin J. Inteligencia Artificial: conceptos, técnicas y aplicaciones. Marcombo, Barcelona-España, p. 284, 1987.
- [Mos96] Moscato P. Optimización heurística y Redes Neuronales, Editorial Paraninfo, España, 1996
- [MS00] Morisio M., Sunderhaft N. Commercial-Off-The-Shelf (COTS): A Survey, Data & Analysis Center for Software - Air Force Research Laboratory, Report SP0700-98-D-4000, USA, 2000.
- [Nor05] Norlund R. Integrating the Rational Unified Process with Managing Successful Programmes. The IBM-Rational White Paper. Disponible en: <http://www-128.ibm.com/developerworks/rational/library/jun05/norlund/index.html>, 2005.
- [NS63] Newell A., Simon H. GPS, a program data simulates human thought. Computer and Thought collection, McGraw-Hill, New York-USA, 1963.
- [Oco84] O'Connor D. Using expert systems to manage change and complexity in manufacturing. Journal of Artificial Intelligence Applications for Business, agosto 1984.
- [Pa82] Papadimitriou H., Combinatorial Optimization: Algorithms and Complexity. Prentice Hall, 1982.
- [Pi02] Pinedo M. Scheduling, Theory, Algorithms and Systems, Prentice Hall, 2^{da} edición., 2002.
- [PMIBOK00] PMI Book. A guide to the Project manangement body of knowledge, edición PMBOK2000, Project Manangement Institute Press, USA, 2000.
- [Pol01] Pollice G. Using the Rational Unified Process for Small Projects: Expanding Upon eXtreme_Programming, Rational Software White Paper TP- 183 3/01, 2001.
- [Pol04] Pollice G. Software engineering for small projects: A RUP centric approach. Addison Wesley, USA, 2004.
- [Pow04] Pow J. Estudio de técnicas basadas en puntos de function para la estimación del esfuerzo en proyectos de software. Revista de Investigación de Sistemas e Informática. Instituto de Investigación de la Facultad de Ingeniería de Sistemas e Informática de la Universidad Nacional Mayor de San Marcos, 1(1): 73-82, Perú, 2004.
- [PT94] Parlot M., Tuytens D. Homogeneous grouping of fuel cans using simulated annealing and tabu search. Annals of operations research, 50:575, 1994.
- [Pre02] Pressman R. Ingeniería de Software. Un enfoque práctico, quinta edición en español. McGraw-Hill, España, 2002.
- [Pro00] Probasco L. The Ten Essentials of RUP The Essence of an Effective Development Process. Rational Software White Paper TP- 177 9/00. Disponible en <http://www.rational.com/worldwide>, 2000.
- [Red04] Reddy A. Project Portfolio Management (PPM): Aligning business and IT. IBM-Rational White paper. Disponible en: <http://www-128.ibm.com/developerworks/rational/library/4779.html>
- [Ri95] Ribeiro C, Prais M A GRASP for a Matrix Decomposition Problem in Traffic Assignment Technical report Catholic University of Rio de Janeiro, Brasil, 1995.

- [Ros05] Rose L. Twelve tips for realistic scheduling in a software development. IBM-Rational White paper. Disponible en: <http://www-128.ibm.com/developerworks/rational/library/jul05/rose/index.html> , 2005.
- [Roy98] Royce W. Software Project Manangement: A Unified Framework. Addison-Wesley, USA; 1998.
- [Roy05] Royce W. Successful software management style: Steering and balance. IBM-Rational White paper. Disponible en: <http://www-128.ibm.com/developerworks/rational/library/mar05/royce/index.html> , 2005.
- [Scha97] Scharage L. Optimization modeling with LINDO. Duxbury Press. USA, 1997.
- [Schn01] Schneider G. Applying Use Cases: A practical guide. Addison-Wesley, USA, 2001.
- [Si99] Sierra J. Planificación Automática. Fondo Editorial de la Escuela Técnica Superior de Ingeniería Informática de la Universidad Autónoma de Madrid – España, 1999.
- [Som92] Sommerville I. Ingeniería de Software (4^{ta} edición) Addison-Wesley, 1992.
- [Ste81] Stefik M. Planning with Constrains. Journal of Artificial Intelligence 16(2): 111-140, 1981.
- [SVD80] Silver E., Vidal R., De Werra D. A tutorial of Heuristic Methods. European Journal of Operational Research, 5(1): 62-74, 1980.
- [Tu01] Tupia, M. Un algoritmo Voraz Adaptativo y Randómico para resolver el problema de la programación de tareas, Tesis para optar el título de Ingeniero Informático, Lima-Perú, Pontificia Universidad Católica del Perú, 2001.
- [VN92] Venugopal V., Narendran T. Cell and package formation in manufacturing systems through simulated annealing: an experimental evaluation. European Journal of Operation Research, 63(3): 409, 1992.
- [We99] Weld D. Recent Advances in AI Planning. Journal of Artificial Intelligence, 20(2): 93-123, 1999.
- [We02] West D. Planning a Project with the IBM - Rational Unified Process. Rational Software publication. Disponible en <http://www.rational.com/products/whitepapers/453.jsp>, 2002.
- [Wi84] Wilkins D. Domain Independent Planning: representation and plan generation. Journal of Artificial Intelligence, 22:1, 1984.
- [YWCH02] Yongpei Guan Y, Wen-Qiang Xiao, Cheung R., Chung-Lu Lin. A multiprocessor task scheduling model for berth allocation heuristic and worst-case analysis. Operations Research Letters, No. 30: 343 – 350, Elsevier Science, 2002.
- [Zah90] Zahniser R. Building software in groups. American Programmer, 3 (7 y 8), USA, jul-ago 1990.
- [Zah94] Zahniser R. Timeboxing for top team performance. Software Development, p. 35-58, USA, mar-1994.

ANEXO

ANEXO A: Detalle de los experimentos numéricos

1. Instancias para N igual a 100 tareas

Instancia	GOLOS0 Muestras	GRASP Muestras	Instancia	GOLOS0 Muestras	GRASP Muestras
100x12_0	222	211	100x25_0	113	113
100x12_1	222	254	100x25_1	117	110
100x12_2	236	193	100x25_2	105	99
100x12_3	207	196	100x25_3	145	140
100x12_4	198	194	100x25_4	120	122
100x12_5	214	171	100x25_5	86	84
100x12_6	188	188	100x25_6	104	99
100x12_7	201	178	100x25_7	87	83
100x12_8	132	172	100x25_8	149	140
100x12_9	201	180	100x25_9	98	93
Promedio	213.3	183.7	Promedio	113.7	108.0
Eficiencia	9.18%		Efficiencia	4.40%	

Tabla A. 1: Instancias $N = 100$, $M = 12$ y 25

Instancia	GOLOS0 Muestras	GRASP Muestras	Instancia	GOLOS0 Muestras	GRASP Muestras
100x37_0	88	87	100x50_0	47	48
100x37_1	71	66	100x50_1	48	52
100x37_2	80	84	100x50_2	77	71
100x37_3	71	68	100x50_3	69	64
100x37_4	86	71	100x50_4	72	52
100x37_5	71	71	100x50_5	50	44
100x37_6	70	84	100x50_6	77	68
100x37_7	71	84	100x50_7	75	70
100x37_8	67	67	100x50_8	60	60
100x37_9	70	71	100x50_9	59	48
Promedio	76.6	74.3	Promedio	66.1	57.8
Eficiencia	2.87%		Efficiencia	2.20%	

Tabla A. 2: Instancias para $N = 100$, $M = 37$ y 50

2. Instancias para N igual a 150 tareas

21

22

Instancia	GOLOS		GRASP	
	Makespan	Makespan	Makespan	Makespan
150x37 0	83	81	79	79
150x37 1	89	88	71	74
150x37 2	91	84	85	91
150x37 3	101	100	84	83
150x37 4	83	81	89	89
150x37 5	83	81	88	87
150x37 6	100	101	87	88
150x37 7	95	94	96	94
150x37 8	72	73	71	74
150x37 9	90	90	84	87
Promedio	88.1	84.1	86.8	87.4
Eficiencia	2.302%		1.793%	

Tabla A. 4: Instancias para $N = 150$, $M = 37$ y 50

3. Instancias para N igual a 200 tareas

Instancia	GOLOS		GRASP	
	Makespan	Makespan	Makespan	Makespan
200x12 0	300	300	414	410
200x12 1	360	361	436	430
200x12 2	300	279	475	458
200x12 3	300	301	444	430
200x12 4	300	298	438	430
200x12 5	300	279	473	465
200x12 6	300	290	472	450
200x12 7	300	304	436	430
200x12 8	300	311	440	430
200x12 9	300	279	450	416
Promedio	300.0	301.0	450.0	417.0
Eficiencia	5.766%		7.834%	

Tabla A. 5: Instancias para $N = 200$, $M = 12$ y 25

Instancia	GOLOS		GRASP	
	Makespan	Makespan	Makespan	Makespan
200x37 0	110	111	77	76
200x37 1	90	84	70	70
200x37 2	110	110	80	80
200x37 3	90	81	70	70
200x37 4	100	90	80	80
200x37 5	100	104	80	80
200x37 6	110	120	80	80
200x37 7	110	100	80	80
200x37 8	90	90	70	70
200x37 9	100	90	80	80
Promedio	100.0	105.0	78.4	78.6
Eficiencia	4.166%		7.755%	

Tabla A. 6: Instancias para $N = 200$, $M = 37$ y 50

4. Instancias para N igual a 250 tareas

Instancia	GOLOS		GRASP	
	Makespan	Makespan	Makespan	Makespan
250x12 0	400	400	1091	1060
250x12 1	471	418	1080	1110
250x12 2	400	400	1000	1000
250x12 3	300	300	1170	1040
250x12 4	411	300	1000	1000
250x12 5	300	200	1090	1000
250x12 6	300	300	1000	1000
250x12 7	300	300	1100	1000
250x12 8	411	300	1100	1000
250x12 9	300	300	1100	1000
Promedio	411.0	377.0	1099.0	1000.0
Eficiencia	8.405%		8.338%	

21 tab012.jpg

22 tab012a.jpg

Tabla A. 7: Instancias para $N = 200$, $M = 12$ y 25

Instancia	GOLOS	GRASP	Instancia	GOLOS	GRASP
200x12_0	110	400	200x25_0	80	83
200x12_1	907	103	200x25_1	100	88
200x12_2	143	120	200x25_2	100	94
200x12_3	132	133	200x25_3	100	80
200x12_4	132	116	200x25_4	84	81
200x12_5	133	123	200x25_5	100	84
200x12_6	120	123	200x25_6	100	83
200x12_7	121	116	200x25_7	100	88
200x12_8	134	124	200x25_8	110	110
200x12_9	120	116	200x25_9	80	80
Promedio	123	118.5	Promedio	95.5	87.1
Eficiencia	4.549%		Eficiencia	1.346%	

Tabla A. 8: Instancias para $N = 250$, $M = 37$ y 50

ANEXO B: Eficiencia de la solución GRASP

1. Calidad de la solución GRASP por el número de iteraciones

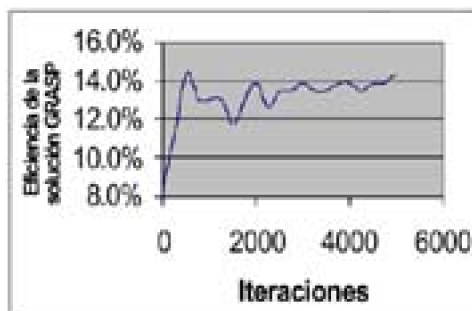


Figura B. 1: Variación de la eficiencia GRASP por iteraciones (100x12)

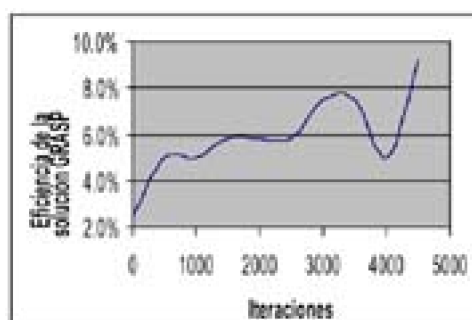


Figura B. 2: Variación de la eficiencia GRASP por iteraciones (150x12)

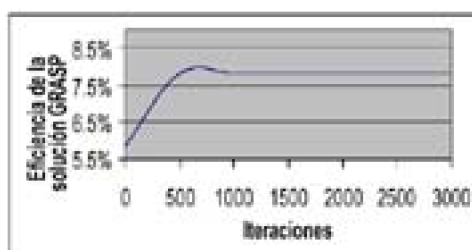


Figura B. 3: Variación de la eficiencia GRASP por iteraciones (200x37)

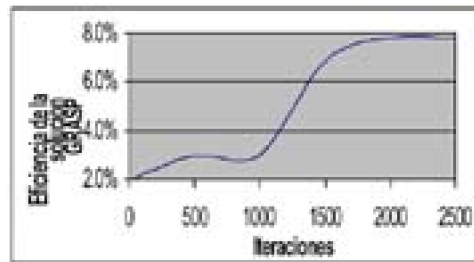


Figura B. 4: Variación de la eficiencia GRASP por iteraciones (250x50)

2. Calidad de la solución GRASP frente al valor de la constante de relajación α

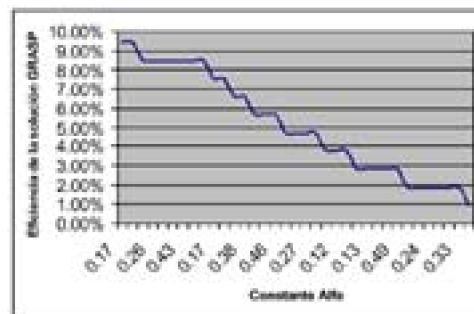


Figura B. 5: Variación de la eficiencia GRASP por el valor de la constante α (100x25)

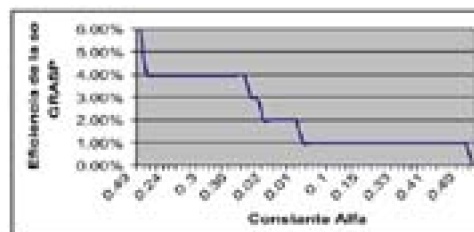


Figura B. 6: Variación de la eficiencia GRASP por el valor de la constante α (150x37)

ANEXO C: Modelos matemáticos usados para pruebas exactas

1. Modelos matemáticos para M igual a 3 máquinas

```

MIN X0
INSTRUCCION          DE
MAXSPAN ES EL EJEMPLO
PRESENTADO EN EL TEXTO
DE LA TESIS
SUBJECT TO
    10 X01 + 11 X21 + 25 X31 =
    10 X41 + 9 X51 + 23 X61 + 27
    X71 + 16 X81 - A1 = 0
    12 X02 + 13 X22 + 24 X32 =
    11 X42 + 10 X52 + 9 X62 + 23
    X72 + 16 X82 - A2 = 0
    13 X03 + 19 X23 + 25 X33 =
    11 X43 + 20 X53 + 8 X63 + 23
    X73 + 12 X83 - A3 = 0
    X0-A1 == 0
    X0-A2 == 0
    X0-A3 == 0
INSTRUCCION          DE
ASIGNACION
    X01 = X12 = X13 = 0
    X21 = X22 = X23 = 0
    X31 = X32 = X33 = 1
    X41 = X42 = X43 = 1
    X51 = X52 = X53 = 1
    X61 = X62 = X63 = 1
    X71 = X72 = X73 = 1
    X81 = X82 = X83 = 1
INSTRUCCION          DE
PROTECCIONES
    X21 = X22 + X23 - (X11 =
    X12 = X13) == 0
    X21 = X22 + X23 - (X41 =
    X42 = X43) == 0
    X21 = X22 + X23 - (X71 =
    X72 = X73) == 0
    X31 = X32 + X33 - (X31 =
    X32 + X33) == 0
    X31 + X32 = X33 - (X61 =
    X62 + X63) == 0
    X41 + X42 = X43 - (X81 =
    X82 + X83) == 0
    X51 + X52 = X53 - (X71 =
    X72 + X73) == 0
    X61 + X62 = X63 - (X61 =
    X62 + X63) == 0
    X71 + X72 = X73 - (X71 =
    X72 + X73) == 0
    X81 + X82 = X83 - (X81 =
    X82 + X83) == 0
END
INT X01
INT X02
INT X03
INT X21
INT X22
INT X23
INT X31
INT X32
INT X33
INT X41
INT X42
INT X43
INT X51
INT X52
INT X53
INT X61
INT X62
INT X63
INT X71
INT X72
INT X73
INT X81
INT X82
INT
X83

```

Figura C. 1: Modelo para $N = 6$, $M = 3$

$X101 + X102 + X103 \leq 0$	INT X40
$X101 + X102 + X103 \leq 0$	INT X31
$X101 + X102 + X103 \leq 0$	INT X12
$X101 + X102 + X103 \leq 0$	INT X53
$X101 + X102 + X103 \leq 0$	INT X01
$X101 + X102 + X103 \leq 0$	INT X02
$X101 + X102 + X103 \leq 0$	INT X03
$X101 + X102 + X103 \leq 0$	INT X71
$X101 + X102 + X103 \leq 0$	INT X72
$X101 + X102 + X103 \leq 0$	INT X73
$X101 + X102 + X103 \leq 0$	INT X01
$X101 + X102 + X103 \leq 0$	INT X02
$X101 + X102 + X103 \leq 0$	INT X03
$X101 + X102 + X103 \leq 0$	INT X91
$X101 + X102 + X103 \leq 0$	INT X02
$X101 + X102 + X103 \leq 0$	INT X03
$X101 + X102 + X103 \leq 0$	INT X101
$X101 + X102 + X103 \leq 0$	INT X102
$X101 + X102 + X103 \leq 0$	INT X103
$X101 + X102 + X103 \leq 0$	INT X111
$X101 + X102 + X103 \leq 0$	INT X112
$X101 + X102 + X103 \leq 0$	INT X113
$X101 + X102 + X103 \leq 0$	INT X121
$X101 + X102 + X103 \leq 0$	INT X122
$X101 + X102 + X103 \leq 0$	INT X123
$X101 + X102 + X103 \leq 0$	INT X131
$X101 + X102 + X103 \leq 0$	INT X132
$X101 + X102 + X103 \leq 0$	INT X133
$X101 + X102 + X103 \leq 0$	INT X141
$X101 + X102 + X103 \leq 0$	INT X142
$X101 + X102 + X103 \leq 0$	INT X143
$X101 + X102 + X103 \leq 0$	INT X151
$X101 + X102 + X103 \leq 0$	INT X152
$X101 + X102 + X103 \leq 0$	INT X153

Figura C. 4: Modelo para $N = 15$, $M = 3$

2. Resultados para los modelos con M igual a 3 máquinas

OBJECTIVE FUNCTION VALUE		
f) 28.00000		
VARIABLE	VALUE	REDUCED COST
X13	1.000000	10.000000
X22	1.000000	0.000000
X31	1.000000	0.000000
X40	1.000000	10.000000
X52	1.000000	0.000000
X63	1.000000	0.000000
X0	28.000000	0.000000
A1	25.000000	0.000000
A2	23.000000	0.000000
A3	28.000000	0.000000
NO. ITERATIONS= 237		
BRANCHES= 40 DETERM= 1.000E-08		

Figura C. 5: Resultados para un modelo con $N = 6$, $M = 3$

OBJECTIVE FUNCTION VALUE		
f) 40.00000		
VARIABLE	VALUE	REDUCED COST
X11	1.000000	10.000000
X21	1.000000	11.000000
X33	1.000000	0.000000
X41	1.000000	10.000000
X51	1.000000	0.000000
X62	1.000000	0.000000
X72	1.000000	0.000000
X83	1.000000	0.000000
X0	40.000000	0.000000
A1	40.000000	0.000000
A2	30.000000	0.000000
A3	35.000000	0.000000
ROW SLACK OR SURPLUS DUAL PRICES		
2)	0.000000	1.000000
3)	0.000000	-1.000000
NO. ITERATIONS= 1094		
BRANCHES= 145 DETERM= 1.000E-08		

Figura C. 6: Resultados para un modelo con $N = 8$, $M = 3$

OBJECTIVE FUNCTION VALUE		
1) 136.0000		
VARIABLE	VALUE	REDUCED COST
X12	1.000000	0.000000
X21	1.000000	0.000000
X33	1.000000	30.000000
X40	1.000000	15.000000
X55	1.000000	85.000000
X61	1.000000	0.000000
X72	1.000000	0.000000
X81	1.000000	0.000000
X92	1.000000	0.000000
X102	1.000000	0.000000
X112	1.000000	0.000000
X121	1.000000	0.000000
X0	120.000000	0.000000
A1	121.000000	0.000000
A2	120.000000	0.000000
A3	120.000000	0.000000
NO. ITERATIONS= 167		
BRANCHES= 17 DETERM = 1.000E 0		

Figura C. 7: Resultados para un modelo con $N = 12$, $M = 3$

OBJECTIVE FUNCTION VALUE		
1) 182.0000		
VARIABLE	VALUE	REDUCED COST
X11	1.000000	0.000000
X21	1.000000	0.000000
X33	1.000000	0.000000
X40	1.000000	0.000000
X51	1.000000	0.000000
X63	1.000000	0.000000
X72	1.000000	57.000000
X81	1.000000	0.000000
X92	1.000000	71.000000
X103	1.000000	0.000000
X112	1.000000	34.000000
X121	1.000000	0.000000
X135	1.000000	0.000000
X140	1.000000	0.000000
X155	1.000000	0.000000
X0	162.000000	0.000000
A1	156.000000	0.000000
A2	162.000000	0.000000
A3	160.000000	0.000000
NO. ITERATIONS= 206		

Figura C. 8: Resultados para un modelo con $N = 15$, $M = 3$

TEXTO COMPLETO EPUB

Descargar texto completo en formato EPUB en la dirección:

http://www.cybertesis.edu.pe/sisbib/2005/tupia_am/epub/tupia_am.epub