Universidad Politécnica de Valencia
Departamento de Sistemas Informáticos y Computación

Nuevos Métodos Meta Heurísticos para la Asignación Eficiente, Optimizada y Robusta de Recursos Limitados

Tesis doctoral

Marzo 2010

Dirigida por:
Dr. Federico Barber Sanchís
Dr. Antonio Lova Ruiz

Presentada por:
Mariamar Cervantes Posada

Nuevos Métodos Meta Heurísticos para la Asignación Eficiente, Optimizada y Robusta de Recursos Limitados

Mariamar Cervantes Posada

Tesis doctoral presentada al
Departamento de Sistemas Informáticos y Computación
para la obtención del grado de

Doctor en Informática

Valencia, 2010

A Juan Carlos

A mi familia

Resumen

Los problemas de optimización y satisfacción de restricciones son extraordinariamente complejos y variados. Al mismo tiempo, son problemas de alto interés, tanto en el aspecto científico-técnico como en el aplicado. Por ello, poder disponer de soluciones algorítmicas eficientes y flexibles supone un alto valor añadido en muy diferentes entornos de aplicación. Entre los problemas más típicos se encuentran los problemas de scheduling o asignación temporal de recursos. Esta clase de problemas implican la ejecución de acciones que requieren recursos cuya disponibilidad está limitada y por tanto deben asignarse de modo eficiente

Dentro de la amplia variedad de los problemas de scheduling, destaca el problema de programación de proyectos con recursos limitados. Dicho problema considera un conjunto de actividades relacionadas entre si mediante relaciones de precedencia, un conjunto de recursos con un límite en su disponibilidad y un conjunto de medidas de desempeño. El objetivo es obtener la mejor manera de asignar dichos recursos a las actividades, de tal manera que se optimice la medida de desempeño.

Se han publicado muchos y diversos trabajos en relación al problema estándar de programación de proyectos con recursos limitados(RCPSP), el cual incluye un único modo de ejecución de las actividades que le conforman, abordando su solución con métodos exactos y métodos aproximados. En cuanto al problema que considera la posibilidad de que cada actividad se ejecute en uno de varios posibles modos (MRCPSP), su estudio no es tan amplio como el del caso anterior.

El objetivo de esta tesis es proponer, diseñar y desarrollar nuevos métodos metaheurísticos para obtener una asignación optimizada de recursos en este complejo problema de scheduling.

Para el caso del RCPSP, hemos seguido un proceso de refinamiento para la propuesta de una heurística y un algoritmo genético utilizando de manera selectiva el método de mejora de programaciones factibles FBI.

En el caso del MRCPSP, hemos diseñado un método de mejora de programaciones factibles y un método de asignación de modos que genera

programaciones factibles en más del 90 % de los casos. Estos métodos han sido incorporados en un algoritmo genético, para el cual hemos diseñado una función de evaluación de los individuos que guía adecuadamente la evolución del algoritmo.

Uno de los supuestos implícitos en estos dos problemas, es que el entorno de desarrollo del proyecto es determinístico. Ello implica que las duraciones de las actividades durante la ejecución del proyecto se mantienen iguales a las planeadas, que la disponibilidad de los recursos no se ve afectada por eventualidades y por lo tanto la programación puede realizarse sin ningún contratiempo. Sin embargo, es claro que dicho supuesto no se cumple en los proyectos que se ejecutan en el mundo real, por lo que se plantea el problema de la generación de programaciones que sean estables en su ejecución.

Para abordar este problema hemos diseñado un algoritmo genético que inserta intervalos de seguridad, y a la vez reserva los recursos necesarios para evitar la propagación de los retrasos primarios a lo largo de la programación. Este algoritmo incorpora una función de evaluación de los individuos que consiste en una medición *ex-ante* de la robustez de las programaciones generadas. Las programaciones generadas se someten a un proceso de simulación con dos escenarios de variabilidad. Se evalúa la robustez de cada programación con dos medidas que hemos propuesto para este caso.

Las propuestas realizadas para esta tipología de problemas se han implementado y sus resultados han sido evaluados mediante la solución de las instancias de las librerías estándar de prueba. La efectividad de estos algoritmos se ha contrastado mediante la comparación con los mejores métodos publicados.

Resum

Els problemes dóptimització i satisfacció de restriccions són extraordinàriament complexos i variats. Al mateix temps, són problemes dált interès, tant en láspecte científic-tècnic com en láplicat. Per això, poder disposar de solucions algorítmiques eficients i flexibles suposa un alt valor afegit en molt diferents entorns dáplicació. Entre els problemes més típics dáquest tipus es troben els problemes de scheduling, que impliquen léxecució dáccions que requerixen recursos la disponibilitat dels quals està limitada i per tant han dássignar-se de manera eficient.

Dins de l'àmplia varietat dels problemes de scheduling destaca el problema de programació de projectes amb recursos limitats, el qual considera un conjunt dáctivitats relacionades entre si per mitjà de relacions de precedència, un conjunt de recursos amb un límit en la seua disponibilitat i un conjunt de mesures de rendiment, i es pregunta quina és la millor manera dássignar els anomenats recursos a les activitats, de tal manera que sóptimitze la mesura del rendiment.

S'han publicat molts i diversos treballs en relació al problema estàndard de programació de projectes amb recursos limitats, el qual inclou una única manera déxecució (RCPSP) de les activitats que el conformen, abordant la seua solució amb mètodes exactes i mètodes aproximats. Quant al problema que considera la possibilitat que cada activitat séxecute en un de diverses possibles maneres (MRCPSP), el seu estudi no és tan ampli com el del cas anterior.

Lóbjectiu désta tesi és obtindre una assignació optimitzada de recursos en aquest complex problema de scheduling. Particularment hem dissenyat i desenvolupat nous mètodes per als problemes mencionats anteriorment. En primer lloc hem seguit un procés de refinació per a la proposta dúna heurística i un algoritme genètic per a resoldre el problema RCPSP utilitzant de manera selectiva el mètode de millora de programacions factibles FBI.

En el cas del MRCPSP hem dissenyat un mètode de millora de programacions factibles i un mètode dássignació de modes que genera programacions factibles en més del 90 % dels casos. Estos mètodes han sigut

incorporats en un algoritme genètic per al qual hem dissenyat una funció dávaluació dels individus que guia adequadament lévolució de lálgoritme.

Un dels supòsits implícits en estos dos problemes és que léntorn de desenvolupament del projecte és deterministic i, per tant, les duracions de les activitats durant léxecució del projecte es mantenen iguals a les planejades, que la disponibilitat dels recursos no es veu afectada per eventualitats i, per tant, la programació pot realitzar-se sense cap contratemps. No obstant això, és clar que el dit supòsit no sácomplís en els projectes que séxecuten en el món real, per la qual cosa es planteja el problema de la generació de programacions que siguen estables en la seua execució.

Per a abordar aquest problema hem dissenyat un algoritme genètic que insereix temps de seguretat, al mateix temps que reserva els recursos necessaris per a evitar la propagació dels retards al llarg de la programació. Aquest algoritme incorpora una funció dávaluació dels individus que consisteix en un mesurament ex-ante de la robustesa de les programacions generades. Dúna altra part es desenrotllen dos mesures de léstabilitat de les programacions que es sotmeten a un procés de simulació amb dos escenaris de variabilitat de la duració de les activitats.

Les propostes realitzades per a esta tipologia de problemes s'han implementat i els seus resultats han sigut avaluats per mitjà de la solució de les instàncies de les llibreries estàndard de prova. Léfectivitat déstos algoritmes s'ha contrastat per mitjà de la comparació amb els millors mètodes publicats.

Abstract

Optimization and constraint satisfaction problems are extraordinarily complex and varied. At the same time, these problems have high interest, as well in the scientific-technical aspect, as in the applied aspect. Therefore, having efficient and flexible algorithmic solutions supposes an extra value in different application environments. Among the most typical problems of this type are scheduling problems, which imply the execution of activities that requiere resources whose availability is limited and therefore must be allocated efficiently.

Among the wide variety of scheduling problems, the problem of resource constrained project scheduling problem is one of the most studied. This problem considers a set of activities that has to be performed in accordance with a set of precedence constraints, a set of scarce resources and a set of performance measures, and decide the best way to allocate those resources to activities in order to optimize the performance measure.

The standard version of the resource constrained project scheduling problem (RCPSP) includes a single execution mode for each activity. This problem has been widely studied in literature and the solving methods proposed are both exact and approximated. As for the problem that considers the possibility that each activity is implemented in one of several possible ways (MRCPSP), his study is not as comprehensive as the previous case.

The objective of this thesis is to obtain an optimized allocation of resources in this complex scheduling problem. Particularly we have designed and developed new approaches to the problems mentioned above. First we followed a refining process for the proposed heuristics and a genetic algorithm to solve the problem RCPSP selectively using the method of improving feasible schedules FBI.

In the case of MRCPSP we have designed an improvement method for feasible schedules and an allocation method that generates mode feasible sequences in more than 90 % of cases. These methods have been incorporated into a genetic algorithm for which we have designed a fitness function that adequately guide the evolution of the algorithm.

One of the assumptions implicit in these two problems is that the project's development environment is deterministic. Therefore the durations of activities during the project execution remain as planned, the availability of resources are not affected by contingencies and the schedule can be executed without changes. However, it is clear that these assumptions are not fulfilled in practice. This is the motivation to solve the problem of generating schedules that are stable in their execution.

To address this problem, we designed a genetic algorithm that inserts time intervals while reserving the necessary resources to prevent the spread of the primary delays along the schedule. This algorithm incorporates an evaluation function of individuals consisting of a measurement *ex-ante* the robustness of the schedules generated. On the other hand develop two measures of the stability of the programs that are subject to a simulation process with two stages of variability of the duration of activities.

All the methods proposed have been implemented and their results have been evaluated by solving the problems of standard libraries. The effectiveness of these algorithms are contrasted by comparing with the best published methods.

Tabla de contenidos

Índice de figuras

Agradecimientos

La gratitud es la memoria del corazón
Anónimo.

No podría terminar esta tesis sin dedicar algunas palabras a aquellas personas que han hecho posible que esta etapa de mi vida llegue a buen término.

A mis directores de Tesis, el Dr. Federico Barber Sanchís y el Dr. Antonio Lova Ruiz, por haberme admitido como su estudiante doctoral. Por todo el tiempo dedicado a discutir las ideas, a organizar mis confusiones y por la paciencia infinita que han tenido conmigo. Por los innumerables consejos y el ánimo permanente que me transmitieron.

A la Dra. Pilar Tormos por compartir generosamente sus conocimientos, experiencia y su tiempo en la mejora continua de este trabajo.

A mis compañeros del *Grupo de Inteligencia Artificial, Planificación y Scheduling* por su apoyo constante, los momentos compartidos y el trabajo en equipo compartido.

Quisiera agradecer a las personas del DSIC por su trato amable, los cafés en la salita y muchos momentos compartidos. En particular quisiera mencionar a mis amigas del laboratorio Laura, Montse, Marlene y otros muy queridos compañeros como Oscar, Antonio, Elena y Luis. El ambiente de trabajo que se ha creado ha contribuido a que esta etapa haya sido feliz y llevadera.

Especial quiero agradecer a mis papás y mis hermanos, que siempre han apoyado mis planes y soy quien soy en gran parte gracias a ellos. A mi esposo Juan Carlos por su apoyo incondicional, por estar siempre a mi lado y por todo su cariño y paciencia sobretodo en los malos días.

Por último agradecer a la Universidad de la Sabana y a la Fundación Carolina por el apoyo económico que brindan a la Investigación.

Mariamar Cervantes P.

1

Introducción

En este capítulo se presentan los rasgos generales del área de investigación que motiva el desarrollo del presente trabajo, el problema de la programación de proyectos con recursos restringidos. Este problema cobra importancia significativa dado el amplio rango de aplicaciones: investigación y desarrollo, construcción, operaciones de mantenimiento o el desarrollo de productos. Una vez descrito el problema, se presenta el marco de trabajo donde se discuten los aspectos más relevantes en el área y la motivación para la realización de esta tesis surge de las líneas de investigación abiertas. Se plantean los objetivos a cumplir durante el desarrollo del presente proyecto y finalmente se hace una descripción del contenido de los capítulos.

1.1. Presentación del Área

Los problemas de satisfacción de restricciones son extraordinariamente complejos y variados. Tienen un amplio interés técnico-científico y aplicado debido a que se encuentran presentes en múltiples ámbitos. Disponer de soluciones algorítmicas eficientes y flexibles supone un alto valor añadido en diversos entornos de aplicación.

En este campo destacan los problemas de scheduling, que implican la programación de tareas que para su ejecución requieren la asignación de recursos limitados [10]. Esta asignación debe hacerse de modo que se optimicen determinados criterios de eficiencia. Los problemas relacionados con la asignación de citas en el sector hospitalario, el establecimiento de horarios en una universidad o la programación de la producción en una industria manufacturera pertenecen a esta categoría.

Debido a la complejidad de estos problemas, el desarrollo de métodos exactos que encuentren la solución óptima en un tiempo razonable se hace muy difícil. Es por ello que los investigadores centran sus esfuerzos en el estudio de nuevos métodos que encuentren soluciones cercanas al óptimo (sub-óptimas) pero de manera eficiente. Entre los métodos más utilizados

por la comunidad académica destacan los sistemas inteligentes, compuestos por algoritmos metaheurísticos, métodos bio-inspirados y los sistemas de razonamiento aproximado.

Dentro de la amplia variedad de los problemas de scheduling destaca el problema de programación de proyectos con recursos limitados, el cual considera un conjunto de actividades relacionadas entre si, un conjunto de recursos y un conjunto de medidas de desempeño, y se pregunta cual es la mejor manera de asignar los recursos a las actividades, de tal manera que la medida de desempeño sea maximizada.

El problema de programación de proyectos es más general que el problema de programación y asignación de recursos como por ejemplo el problema de programación de máquinas (Blazewicz [11]) en el sentido en que los trabajos tienen relaciones de precedencia más complejas y adicionalmente pueden requerir de varios recursos para su ejecución. Por lo tanto los problemas de programación de máquinas son casos especiales de la programación de proyectos.

Debido a ello, los problemas de programación de proyectos capturan muchos de los aspectos prácticos que existen en diversas industrias como la construcción, la ingeniería, sistemas justo a tiempo o el desarrollo de software. [16]. Adicionalmente se ha demostrado[11] que pertenece a la clase de problemas NP-duros y se comprobó que encontrar la solución óptima requiere tiempos muy altos para proyectos de 30 o más actividades, incluso cuando hay un único modo de ejecución.

1.2. Gestión de Proyectos

La definición de **proyecto** ha sido ampliamente discutida a través de los años, se matiza de un autor a otro y de un contexto a otro. Inicialmente el término estaba asociado a las grandes obras de construcción civil o a las campañas militares. El Project Management Institute[79] ha definido proyecto como "*Un esfuerzo temporal realizado para crear un producto, servicio o resultado único*"; y entre los autores es común encontrar definiciones similares a "*la realización de una actividad compleja susceptible de descomponerse en una serie de tareas o actividades interdependientes entre si en cuanto a su orden de ejecución*"[83]

En términos generales un **proyecto** se puede definir como un conjunto de actividades interdependientes que se desarrollan en un periodo determinado con el propósito de lograr un objetivo según ciertos requerimientos incluidas restricciones de tiempo, costo y recursos.

En todo proyecto existen tres factores fundamentales: Tiempo, Coste y Calidad, que constituyen lo que se ha llamado el triángulo del proyecto, que se representa en la Figura 1.1.

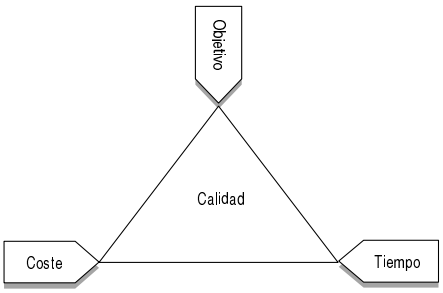


Figura 1.1. Triángulo del Proyecto

Todo proyecto está sujeto al **tiempo**, es decir tienen una fecha límite en la que el proyecto debe estar concluido. Además, es posible que el proyecto disponga de una serie de hitos intermedios por cumplir.

El **coste** está asociado al factor humano, maquinaria, material, uso de instalaciones u otros elementos que al final se traducen en un presupuesto económico. Para todos los proyectos el coste supone una limitación.

La cantidad de tiempo dedicado a las tareas individuales determina la **calidad** global del proyecto. Algunas tareas pueden requerir una cantidad dada de tiempo para ser completadas adecuadamente, pero con más tiempo podrían ser completadas excepcionalmente. A lo largo de un proyecto la calidad puede tener un impacto significativo en el tiempo y en el coste (o viceversa).

Finalmente, el **objetivo** es completar el proyecto según las restricciones de tiempo, coste y calidad establecidas por el cliente.

Tanto en áreas de producción como en servicios, cada vez más las compañías utilizan una organización basada en proyectos dentro de un rango muy amplio de aplicaciones: investigación y desarrollo, desarrollo de software, construcción, infraestructura pública, operaciones de mantenimiento, desarrollo de productos, etc.

Para manejar de manera adecuada los tres factores de un proyecto surge la gestión de proyectos (*project management*) que es la disciplina de organizar y administrar recursos de manera tal que se pueda culminar todo el trabajo requerido en el proyecto dentro del alcance, el tiempo, y coste definidos.

Tal como se ve en la Figura 1.2, el ciclo de vida del proyecto consta de cuatro etapas.

La *Planificación del proyecto* consiste en definir los objetivos, el presupuesto, las actividades y en estimar sus duraciones aproximadas y los recursos que requieren. Las técnicas de planificación se ocupan de estructurar las tareas a realizar dentro del proyecto, definiendo la duración y el orden de ejecución de las mismas.

Durante la *Programación del proyecto* se establecen las secuencias en las que se especifica el tiempo de inicio para cada actividad teniendo en cuenta

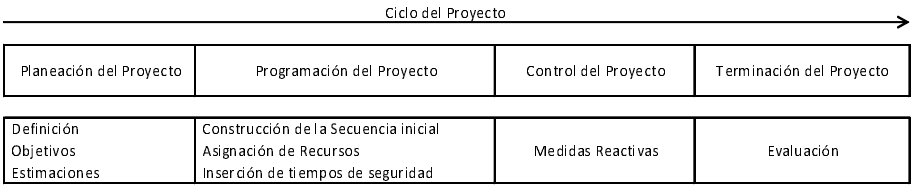


Figura 1.2. Ciclo de Vida del Proyecto

la restricción de recursos y las relaciones de precedencia entre las actividades. Dicha secuencia es la base que se utiliza en la ejecución del proyecto. Así mismo, permite visualizar el proyecto de inicio a fin y es una herramienta valiosa de comunicación y planificación interna y externa.

Una vez que el proyecto y sus componentes han sido definidos de la manera adecuada, el proyecto se pone en ejecución y se inicia con el la *fase de control* del trabajo realizado. Se analizan las diferencias entre lo programado y lo ejecutado y de ser necesario se ponen en marcha las medidas correctivas que sean necesarias.

Una vez finalizado el proyecto debe hacerse un *Análisis y Evaluación* de las diferencias presentadas entre lo planeado y lo ejecutado y la manera en que se resolvieron estas diferencias en la fase de control. La satisfacción del cliente es el objetivo último de la gestión de proyecto. Esta fase proporciona nuevos factores a la hora de planificar un proyecto similar, estimar mejor los tiempos de duración o los recursos requeridos, proporcionando un aprendizaje importante a los responsables de la planificación y control.

Demeulemeester & Herroelen [27] hacen una revisión detallada del ciclo de vida del proyecto. El trabajo desarrollado en esta tesis se enmarca en la segunda etapa del ciclo de vida del proyecto.

1.3. Descripción del problema

Los problemas de programación de proyectos están constituidos por actividades, recursos, relaciones de precedencia y función objetivo[58], tal como se describe a continuación.

Un proyecto consiste de un número de **actividades**, también conocidas como trabajos, operaciones o tareas. Para completar el proyecto exitosamente, cada actividad puede ser procesada en uno de varios posibles modos. Cada modo representa una manera diferente de realizar la actividad que está siendo considerada. El modo determina la duración de la actividad, los requerimientos de los diferentes tipos de recursos, los posibles flujos (positivos o negativos) de dinero y cualquier otra característica asociada con la actividad.

Entre las actividades del proyecto se definen las **relaciones de precedencia** cuando se determina el orden en que deben ejecutarse. Las relaciones de precedencia se dan por razones tecnológicas o de diseño del

proceso. Para ello el proyecto se representa como un grafo dirigido donde las actividades están representadas en un nodo y las relaciones de precedencia entre actividades están en los arcos.

Como se mencionó, cada actividad debe hacer uso de un cierto número de **recursos** para su ejecución. Los recursos utilizados por las actividades se clasifican en cuatro categorías: renovables, no renovables, doblemente limitados y recursos parcialmente (no) renovables.

Los Recursos Renovables como las horas hombre, máquinas, herramientas, equipos, espacio están disponibles periodo a periodo, es decir, la cantidad disponible se renueva de un periodo a otro. Los Recursos no renovables como dinero, materias primas o energía, tienen un total disponible durante todo el proyecto y se van consumiendo conforme se ejecutan las actividades. Los recursos doblemente limitados están limitados tanto por periodo como el total disponible en todo el proyecto. Más recientemente se ha introducido el concepto de Recursos Parcialmente (No) Renovables que hace referencia a recursos cuya disponibilidad se renueva en intervalos específicos de tiempo. Este tipo de recursos pueden ser vistos como un concepto genérico de todos los tipos de recursos.

La **disponibilidad** de un recurso es la cantidad de dicho recurso que puede ser utilizada en un momento dado y que puede ser constante o variable, y el **requerimiento** de una actividad es la cantidad de recurso que se necesita para que pueda ser procesada.

La programación de proyectos puede tener diversos **objetivos** y con base en ellos se mide la bondad o calidad de una solución. Los objetivos típicamente considerados son los siguientes:

- Minimizar la Duración del proyecto: es, sin lugar a dudas, la medida más aplicada en el dominio de la programación de proyectos. La duración está definida como el intervalo de tiempo entre el comienzo y el fin del proyecto. Como el comienzo del proyecto es usualmente asumido en el tiempo $t = 0$, minimizar la duración es equivalente a minimizar el máximo de los tiempos de finalización de todas las actividades.
- Maximizar el Valor Actual Neto del proyecto: cuando en proyectos de gran envergadura y a largo plazo están presentes cantidades significativas de flujo de dinero, en forma de gastos para iniciar las actividades y pagos para completar las partes del proyecto; el valor actual neto (VAN) es un criterio adecuado para medir la optimalidad del proyecto. Este criterio genera una ruta crítica de coste y no la ruta crítica de tiempos generada cuando se minimiza la duración.
- Maximizar la Calidad del proyecto: Este objetivo es muy importante para los directores de proyecto. La calidad de un proyecto está dada porque éste se haga dentro de los plazos planeados, cumpla con el presupuesto y que el cliente quede satisfecho con el producto. La formulación de este problema se ha centrado en minimizar la desviación de los plazos y el presupuesto establecido debido a actividades que deben ser preprocesadas.

- Minimizar el Coste del proyecto: Este objetivo ha atraído mucha atención de los investigadores debido a su relevancia práctica. Se puede minimizar el coste de actividades ya que distintas maneras de desarrollar una actividad resulta en distintos costes directos, los cuales deben ser minimizados (intercambio coste-duración). Por otra parte puede plantearse minimizar el coste de los recursos que está determinado por la programación de las actividades, la cual influencia el coste indirectamente a través de los recursos.

En esta tesis abordamos el problema de programación de proyecto con recursos limitados cuyo objetivo es minimizar la duración del proyecto. En primer lugar estudiamos su versión estándar, que es aquella donde las actividades tiene un único modo de ejecución. Las actividades están relacionadas entre si por relaciones de precedencia inicio-fin sin holguras y utilizan para su ejecución un conjunto de recursos renovables.

Así mismo estudiamos el caso en el que las actividades se ejecutan en uno de múltiples modos de ejecución, que es una extensión del problema anterior. En este problema las actividades utilizan recursos renovables y no renovables para su ejecución.

Debido a los eventos imprevistos que surgen durante la ejecución del proyecto, la programación no puede ejecutarse completamente tal como fue planeada. Por ello nos centramos en el problema de la generación de programaciones que sean estables durante su ejecución. Es decir, aquellas programaciones que son poco sensibles a los eventos externos y por lo tanto las diferencias entre la programación planeada y la que se realiza durante la ejecución sean las menores posibles.

1.4. Marco de Trabajo

Gran parte de la investigación realizada en el área de la programación de proyectos con recursos limitados se concentra en minimizar la duración del proyecto en condiciones determinísticas. Las actividades están sujetas a relaciones de precedencia inicio-fin sin holguras. Puede ser el caso en que las actividades tienen un único modo de realizarse y todos los recursos son renovables o el caso más general de múltiples modos de ejecución y recursos renovables y no renovables.

1.4.1. Programación de Proyectos con Recursos Limitados y Único Modo de Ejecución (RCPSP)

El problema de programación de proyectos con recursos restringidos y un único modo de ejecución de las actividades (RCPSP) es una generalización del problema de Job Shop [11] y ha atraído la atención de un gran número de investigadores.

Con el desarrollo de procedimientos heurísticos surgió la necesidad de generar instancias para evaluarlos y compararlos. Actualmente, los investigadores utilizan la librería estándar de pruebas PSPLIB [59] que cuenta con problemas de tamaño pequeño (30 actividades), mediano (60 actividades) y grande (120 actividades).

Los problemas de 30 actividades pueden ser resueltos por métodos exactos y por tanto se conocen sus soluciones óptimas. Las revisiones más recientes sobre métodos exactos son los trabajos de Kolisch y Padman [58] y Demeulemeester y Herroelen [27].

Debido a que el RCPSP es un problema NP-duro [11] no es posible resolver todas las instancias de manera óptima. Los métodos exactos no pueden tratar con problemas de tamaño mediano o grande debido a la cantidad de variables y restricciones generadas y a la amplitud del espacio de búsqueda. La obtención de secuencias de buena calidad en un tiempo razonable solamente puede lograrse mediante el uso de métodos aproximados desarrollados para tal fin.

Kolisch y Hartmann [56], [40] [57] hacen una revisión de estos métodos. Los algoritmos genéticos son el enfoque que obtiene los mejores resultados en la resolución de este problema. Se utiliza una codificación de los individuos basada en una lista de actividades y se utilizan los esquemas serie y paralelo para generar la programación del proyecto. Todos los algoritmos con mejor desempeño utilizan el método de mejora Forward-Backward (FBI) desarrollado por Tormos y Lova [94] el cual está basado en las listas de actividades y el esquema serie.

Debido a la gran actividad en este campo, el desarrollo de métodos eficientes computacionalmente y que obtengan un desempeño comparable con los mejores algoritmos reportados hasta el momento constituyen un avance en esta área.

1.4.2. Programación de Proyectos con Recursos Limitados y Múltiples Modos de Ejecución (MRCPSP)

Cuando las actividades pueden ser ejecutadas uno de múltiples modos (MRCPSP) se genera una decisión adicional para crear la programación. Es necesario determinar el modo de ejecución para cada actividad y así asignar un tiempo de inicialización o finalización de las actividades, de tal manera que se cumplan las relaciones de precedencia, no se sobrepase la disponibilidad de los recursos y la duración del proyecto se minimice.

Kolisch [55] demostró que la asignación de modos es un problema NP-completo cuando existen al menos dos recursos no renovables y el problema multimodo en sí mismo es NP-duro. Los procedimientos de optimización más actuales son incapaces de resolver proyectos con recursos altamente restringidos de más de 20 actividades y tres modos de ejecución por actividad [88] por tanto, en la práctica los métodos aproximados son los únicos capaces de generar secuencias factibles y cercanas al óptimo.

Debido a la complejidad del problema, los mejores algoritmos incluyen siempre el procedimiento de preproceso propuesto por Sprecher et al[89] en el cual se excluyen los modos no ejecutables e ineficientes. La idea subyacente es excluir modos y/o recursos no renovables de los datos de entrada. En la actualidad, la mayoría de los métodos de solución aproximados utilizan una asignación aleatoria de los modos como punto de partida del algoritmo.

Los algoritmos aproximados están basados en los esquemas serie y paralelo, que construyen la programación teniendo en cuenta el modo asignado a la actividad. Por tanto la programación es factible con respecto a los recursos no renovables si la asignación de modos es factible.

La librería de pruebas PSPLIB tiene una sección dedicada a este tipo de problemas, la cual consiste en problemas de 10, 12, 14, 16, 18, 20 y 30 actividades, con 2 recursos renovables y 2 no renovables. Se conocen las soluciones óptimas para los conjuntos de menos de 30 actividades. Los algoritmos desarrollados se prueban también con el conjunto de Boctor [12], el cual consta de dos conjuntos de problemas, uno con 50 y el otro con 100 actividades con 1, 2 o 4 recursos renovables.

Los algoritmos con mejor desempeño son métodos evolutivos (Algoritmos Genéticos y Scatter Search). Un aspecto importante que se debe tener en cuenta, es que no es posible generar en todos los casos soluciones factibles con respecto a los recursos no renovables, y por ello se hace necesaria una función de evaluación que penalice dicha infactibilidad.

En este problema, a diferencia del RCPSP, no existe un método de mejora que aproveche las holguras libres entre las actividades ni la posibilidad de cambiar el modo de ejecución de las actividades. Por lo tanto un desarrollo en esta línea sería una aportación a los métodos que se desarrollen en el futuro.

1.4.3. Generación de Programaciones Robustas

Uno de los supuestos que se mantiene en estos dos problemas, es que el entorno de desarrollo del proyecto es determinístico. Ello implica que las duraciones de las actividades durante la ejecución del proyecto se mantienen iguales a las planeadas y que la disponibilidad de los recursos no se ve afectada por eventualidades. Por lo tanto la programación puede realizarse sin ningún contratiempo.

Sin embargo, es claro que dicho supuesto no se cumple en los proyectos que se ejecutan en el mundo real. Las máquinas se averían, las actividades pueden durar más de lo planeado debido a problemas en la cadena de abastecimiento o un subcontratista no cumple con su plazo de entrega. Esta situación plantea el problema de la generación de programaciones que sean estables en su ejecución.

En términos generales una programación se considera como *robusta* cuando es poco sensible a las variaciones que pueden ocurrir durante la ejecución del proyecto. Es decir,cuando puede absorber las disrupciones sin que se afecte la programación de las actividades hecha previamente.

En este problema existen dos aspectos que deben ser abordados y que están intrínsecamente relacionados. De un lado la manera de generar programaciones robustas y de otra, cómo medir dicha robustez. La manera más común de generar programaciones robustas es mediante la inclusión de tiempos de seguridad (buffers) entre actividades. Estos tiempos de seguridad absorben los retrasos primarios y evitan su propagación por la programación.

Por lo general se considera una secuencia inicial sobre la cual se insertan los buffers par protegerla de eventos inesperados. Cualquier método de solución para el RCPSP usando la duración media de las actividades sirve para construir la secuencia inicial. Sobre la duración inicial se considera un plazo adicional y se determina la finalización prevista (δ_n).

Las heurísticas existentes para la inserción de buffers, insertan más tiempo de seguridad a aquellas actividades que tienen más impacto en la estabilidad de la programación. El impacto puede estar asociado al costo de retraso de la actividad, a su duración o a lo crítico que sea su tiempo de inicio. En este punto es necesario comparar las diversas soluciones generadas, para este propósito existen diversas medidas de la robustez de dichas programaciones antes de ser ejecutadas. Estas medidas se basan en la holgura de las actividades.

Para comprobar la estabilidad de las soluciones se someten a un proceso de simulación el cual genera un escenario de ejecución; la programación se ejecuta bajo estas condiciones y se evalúa su robustez. Tanto el método proactivo como las decisiones reactivas que se toman a medida que se ejecuta el proyecto tienen una gran influencia en la secuencia realizada.

En este aspecto, es relevante el desarrollo de métodos computacionalmente eficientes en la inserción de tiempos de seguridad así como el diseño y evaluación de medidas de la robustez de las programaciones; tanto medidas *ex-ante* como posteriores a la simulación de la secuencia y de la ejecución global del proyecto.

1.5. Motivación

Una vez presentada el área de trabajo se evidencia la importancia de contar con soluciones algorítmicas eficientes y flexibles, las cuales suponen un alto valor añadido en el ámbito de la programación de operaciones (*scheduling*). Dentro de los problemas de scheduling, uno de los más estudiados y de los más ampliamente difundidos es el problema de Job-shop[77].

El problema de Programación de Proyectos con Recursos Limitados es una generalización de dicho problema [11] en el sentido en que los trabajos tienen relaciones de precedencia más complejas y adicionalmente pueden requerir de varios recursos para su ejecución. Dado que pertenecen a la clase de problemas NP-duros [11] es imprescindible recurrir al uso de métodos aproximados para su solución. En general, para un problema de tamaño real no existe ningún método que permita certificar si una solución es la óptima.

En la práctica, el hecho de minimizar la duración de un proyecto genera ventajas como tener un presupuesto más bajo debido a la mejor utilización de los recursos y a la reducción de tiempos ociosos. Estos son factores clave que hace que un cliente firme la ejecución del proyecto con un determinado proveedor. Por ello este trabajo se centra en los problemas de programación de proyectos con dicha función objetivo (RCPSP y MRCPSP).

Gracias a la librería estándar de pruebas PSPLIB, es posible comparar los resultados obtenidos por los diversos algoritmos ya que permiten el uso de un banco de pruebas común. Este hecho promueve una competencia por encontrar soluciones de mejor calidad (menor duración de la secuencia) y se ha llegado a un punto tal en que no es fácil conseguir mejoras frente a los métodos publicados hasta el momento.

Es una realidad que una vez llevada a la ejecución la programación generada por los métodos enunciados anteriormente se verá expuesta a múltiples incidencias: retraso en la entrega de los proveedores, cambio en la disponibilidad de recursos o que la estimación del tiempo de ejecución de una actividad fue errónea.

Por ello es de interés la generación de secuencias que sean estables ante las eventualidades que puedan surgir en la ejecución del plan. Para aumentar la estabilidad de una programación determinada se hace necesaria la introducción de tiempos de seguridad, lo que conlleva a que si se quiere una secuencia robusta no se tendrá una secuencia de duración mínima. Hay un intercambio entre robustez y duración.

1.6. Objetivos y Principales Contribuciones

La presente tesis tiene como objetivo principal el desarrollo de nuevos enfoques característicos para la resolución de problemas de programación de proyectos con recursos restringidos (RCPSP y MRCPSP). Así mismo, los subobjetivos parciales el logro de este trabajo se detallan a continuación:

a. Desarrollo de técnicas de asignación optimizada de recursos, con especial atención a la eficiencia computacional, a fin de poder abordar la resolución de problemas complejos en el área.
b. Desarrollo de técnicas de asignación orientadas hacia la robustez de las soluciones, tal que sean capaces que mantener su optimalidad ante incidencias en su ejecución en un entorno dinámico y parcialmente conocido.
c. Evaluación y contraste de los métodos desarrollados, mediante su aplicación a la resolución de problemas del banco de pruebas estándares PSPLIB.

Las contribuciones de esta tesis en cuanto al problema RCPSP consisten en el diseño e implementación de una heurística basada en reglas de prioridad que hace un uso selectivo del método de mejora FBI. Las reglas con mejor desempeño se incorporan en un algoritmo genético que utiliza el método de

mejora FBI según la profundidad de la exploración del espacio de solución y la calidad de la solución base. Los métodos propuestos se evalúan utilizando la librería PSPLIB. La heurística de múltiples pasadas es competitiva con métodos metaheurísticos y el algoritmo genético está entre los cinco mejores métodos propuestos.

En cuanto al problema MRCPSP las contribuciones abordan diferentes aspectos. Se diseña un método de asignación de modos el cual genera soluciones factibles en más del 95 % de los casos siendo computacionalmente eficiente. Se extiende el método de mejora FBI que aprovecha las holguras de las actividades y la posibilidad de cambio de modo de ejecución para compactar la programación. Estos dos métodos se incorporan en un algoritmo genético, para el cual se diseña una función de evaluación de los individuos que supera las desventajas de otras presentadas previamente en la literatura. Así mismo se desarrolla un operador de mutación masiva con el objetivo de factibilizar las soluciones que no cumplen con la restricción de los los recursos no renovables luego de ejecutarse los operadores genéticos.

Finalmente, en el tema de la generación de programaciones robustas hay dos contribuciones principales. Por un lado se diseña por primera vez un algoritmo genético para la asignación de los tiempos de buffer a las actividades. Para ello se extiende la representación de lista de actividades y se presenta una función para la medición *ex-ante* de la robustez de las programaciones generadas. La robustez de las programaciones se prueban mediante un proceso de simulación. Se diseñan dos medidas de la estabilidad de las programaciones que son evaluadas.

La validación de este algoritmo se hace mediante el uso de la librería de pruebas PSPLIB en el caso RCPSP. Debido a que no es posible la comparación con otros autores, se desarrolla un método aleatorio de asignación de buffers. El algoritmo genético tiene resultados consistentemente mejores.

1.7. Estructura

En el Capítulo 2 se hace una introducción a la familia de problemas de la programación de proyectos. Así mismo, se presenta la formulación matemática de cada uno de los problemas abordados en esta tesis y se revisan los diferentes métodos de solución que están publicados en la literatura. Al final de este capítulo se presentan las librerías estándar de prueba que se utilizan para evaluar los métodos propuestos.

En el Capítulo 3 se propone una heurística y un algoritmo genético para la solución del problema RCPSP. La heurística es un método aleatorizado basado en reglas de prioridad y el uso selectivo del método de mejora FBI. Luego se propone una familia de algoritmos genéticos, donde se estudian diversas funciones para cada uno de los operadores genéticos. Finalmente, tanto la heurística como los mejores algoritmos genéticos desarrollados se comparan con los métodos reportados en la literatura.

En el Capítulo 4 se aborda el problema MRCPSP. Inicialmente se propone un método de mejora de programaciones factibles. Este método aprovecha la posibilidad de cambio de modo de las actividades para compactar la programación inicial, siempre y cuando se cumplan las restricciones de recursos. Adicionalmente se diseña un método para la selección de modos que genera asignaciones factibles en un alto porcentaje. Estos elementos se incorporan en un algoritmo genético, para el cual se diseña una función de evaluación de los individuos que guía de manera eficiente la búsqueda en el espacio de solución. Cada uno de estos elementos es evaluado mediante la solución de problemas estándar utilizados en la literatura y finalmente se compara el resultado del algoritmo con los problemas de dos conjuntos de datos estándar.

En el Capítulo 5 se propone un algoritmo genético como método proactivo de generación de programaciones robustas. El objetivo de este procedimiento es la inserción de buffers de seguridad frente a ciertas actividades, de tal manera que la programación resultante sea capaz de absorber los imprevistos que surgen durante la ejecución. Para ello se ha diseñado una medida *ex-ante* de la robustez de la solución. Para comprobar la efectividad de dichas soluciones se simula el proyecto en dos escenarios de variabilidad de las actividades. También se presentan dos métricas de la robustez de la solución. El estudio computacional se hace con la librería de pruebas estándar PSPLIB en el caso de modo único y finalmente se analiza el impacto de las características del proyecto frente a la robustez.

En el Capítulo 6 se presentan las conclusiones y posibles líneas de trabajo futuro.

2

Revisión General del Estado del Arte

2.1. Introducción

Los problemas de programación de proyectos constituyen una familia de diversos problemas según su objetivo, los tipos de recursos disponibles y la topología de la red.

Esta familia de problema presenta distintas variantes. El problema básico consiste en minimizar la duración del proyecto cuando las actividades que lo componen no pueden interrumpir su ejecución y están sujetas exclusivamente a relaciones de precedencia de tipo Fin-Inicio. Las actividades utilizan para su ejecución un conjunto de recursos renovables con disponibilidad limitada y constante a lo largo de todo el proyecto. Las actividades tienen un único modo de ejecución, con una duración determinada y un consumo dado de recursos. También se suele hacer referencia a él como la versión estándar del problema (RCPSP) o la versión de "modo único".

Otra clase de problema se genera cuando las actividades presentan distintos modos de ejecución. Es decir, cada una de las actividades puede presentar distintas posibilidades o modos diferentes de ejecución. Un modo no es más que una forma de ejecutar la actividad, que lleva asociada una determinada duración y un determinado consumo de recursos. A esta versión se hace referencia como la versión "multi-modo" del problema (MRCPSP), en la que pueden aparecer además recursos no renovables y doblemente limitados.

Existen otras muchas variantes, como puede ser el permitir interrumpir la ejecución de las actividades, incluir otros tipos de relaciones de precedencia además del tipo Fin-Inicio, establecer fechas obligadas máximas y/o mínimas para las actividades, retrasos máximos y/o mínimos entre actividades, disponibilidades variables de los recursos a lo largo del tiempo, utilización de recursos de los tres tipos, etc. Además se pueden establecer objetivos diferentes como por ejemplo maximizar el valor neto del proyecto, nivelar la demanda de los recursos, etc.

En este trabajo nos centramos en dos de los problemas: el problema de la programación de proyectos con recursos limitados (RCPSP) y en su

extensión a múltiples modos de ejecución (MRCPSP). Estos problemas tienen un gran interés práctico ya que son una generalización del problema de Job-shop, ampliamente estudiado en los problemas relacionados con la producción. Debido a que estos problemas manejan el supuesto de un entorno de ejecución estático, que no se cumple en las situaciones de la vida real, abordamos el problema de generar una programación que se mantenga cuando en la ejecución del proyecto hay variabilidad en la duración de las actividades.

Para hacer la revisión del estado del arte este capítulo se estructura de la siguiente manera: en la sección 2.2 presentamos la formulación matemática de los tres problemas. Los métodos de solución del RCPSP y MRCPSP se revisan en la sección 2.3 y los que abordan el problema de la generación de secuencias robustas se presentan en la sección 2.4. Debido a la importancia de los Algoritmos Genéticos como método de solución en el área de programación de proyectos, la sección 2.5 está dedicada a ellos. Finalmente en la sección 2.7 se presentan las conclusiones de este capítulo

2.2. Formulación de los problemas

2.2.1. Programación de proyectos con recursos limitados y modo único RCPSP

El problema de la Programación de Proyectos con Recursos Limitados y un modo único de ejecución de las actividades, ha sido ampliamente estudiado en la literatura, es un problema combinatorio clásico que aparece en las líneas de manufactura y producción, en particular en líneas de ensamble o sistemas de producción Justo a Tiempo.

La formulación de modelos matemáticos ha sido abordada por diversos autores. Los más difundidos en la literatura son Pritsker et al.[81], Kaplan[47], Alvarez-Valdes & Tamarit[6] y Mingozzi et al.[72].

A continuación se presenta, a manera de ejemplo la formulación de Pritsker et al. [81]. En este modelo las variables indican si una actividad finaliza o no en su ejecución en un determinado instante. Así pues se definen las **variables**:

$$F_{jt} = \begin{cases} 1 & \text{si la actividad } j \text{ finaliza en el instante } t, \\ 0 & \text{en otro caso.} \end{cases}$$

F_{jt} solo existe en los casos en que $EF_j \leq t \leq LF_j$ donde EF_j es el tiempo de inicio más temprano y LF_j el tiempo más tardío de inicio de la actividad j.

Antes de construir el modelo, es necesario establecer una cota superior del tiempo de ejecución del proyecto. Esta cota se denota como T. Cuanto menor sea el valor de T, es decir, cuanto más ajustada sea la cota, menor tamaño tendrá el modelo resultante. El método más sencillo para estimar T es sumar la duración de todas las actividades que componen el proyecto:

$$T = \sum_{\forall j \in J} d_j$$

La formulación del problema es la siguiente:

$$Min \sum_{t=\mathrm{ES}_{n+1}}^{\mathrm{LF}_{n+1}} t\ F_{n+1} \tag{2.1}$$

sujeto a:

$$\sum_{t=ES_j}^{LS_j} F_j = 1 \qquad \forall j \in J \tag{2.2}$$

$$\sum_{\forall j \in J} \sum_{q=t}^{t+d_j-1} r_{jk} F_{jq} \leq R_k \qquad \forall k \in K,\ \ \forall t\ /\ 1 \leq t \leq T \tag{2.3}$$

$$\sum_{t=ES_i}^{LS_i} t\ F_{it} \leq \sum_{t=ES_j}^{LS_j} (t - d_j) F_{jt} \qquad \forall (i,j) \in E \tag{2.4}$$

$$\tag{2.5}$$

El objetivo es minimizar la duración del proyecto determinado por el instante de finalización de la actividad ficticia $n + 1$. En cuanto a las restricciones, el grupo 2.2 asegura que se ejecuten todas las actividades y que no sean interrumpidas una vez ha empezado su ejecución. La ecuación 2.3 exige que se cumplan las restricciones relativas al consumo de recursos y 2.4 obliga a que se cumplan las relaciones de precedencia entre las actividades.

2.2.2. Programación de proyectos con recursos limitados múltiples modos MRCPSP

Desde el comienzo de la gestión de proyectos, los desarrolladores del método CPM reconocieron que en la práctica la mayoría de las actividades pueden desarrollarse con una duración mayor o menor según los recursos que se asignan a ellas, por ejemplo mano de obra, maquinaria, dinero, etc. Fue propuesto formalmente en la literatura por Elmaghraby en 1977 [29].

Este problema es relevante en sistemas de manufactura flexible y otros ambientes de programación de operaciones donde se permiten diferentes rutas alternas para los trabajos y diferentes máquinas. Las herramientas y los equipos tienen diferentes requerimientos de material y manejo.

El MRCPSP estándar involucra la selección de un modo de ejecución para cada actividad y la asignación de un tiempo de inicialización o finalización de las actividades, de tal manera que se cumplan las relaciones de precedencia, no se sobrepase la disponibilidad de los recursos y la duración del proyecto se minimice.

Existen dos decisiones para determinar la secuencia con la duración óptima del proyecto: la selección de un modo de ejecución para cada actividad (*asignación de modo*)y la determinación del comienzo o del fin de cada actividad de tal manera que las restricciones de precedencia y de recursos sean respetadas y que la duración del proyecto sea mínima.

En 1982 Talbot [92] introdujo un modelo de programación binaria, donde las variables de decisión representan el tiempo de inicio de cada actividad y el modo en que será ejecutada:

$$F_{jmt} = \begin{cases} 1 & \text{si la actividad } j \text{ se ejecuta en modo } m \text{ y finaliza en el instante } t, \\ 0 & \text{en otro caso.} \end{cases}$$

En este caso la cota superior del proyecto T, se obtiene sumando las duraciones máximas de todas las actividades del proyecto. En cada caso, el número de modos en el cual cada actividad i puede ser ejecutada se denota como M_i. La duración de la actividad i cuando se ejecuta en el modo m se denota como d_{im}.

Se utilizan múltiples recursos renovables k (k=1, 2,. . . ,K) cuya disponibilidad es constante a través del tiempo. La disponibilidad del recurso renovable k en el periodo t se denota como R_k^ρ.

Los recursos no renovables tienen una disponibilidad determinada para todo el horizonte de proyecto. En el MRCPSP se utilizan múltiples recursos no renovables k (k=1, 2,. . . ,K) cuya disponibilidad en el horizonte T del proyecto es R_k^υ.

$$\text{Minimizar} \sum_{t=ES_{n+1}}^{LS_{n+1}} tF_{(n+1)1t} \tag{2.6}$$

sujeto a:

$$\sum_{m=1}^{M_j} \sum_{t=ES_j}^{LS_j} F_{jmt} = 1 \qquad \forall j \in J \tag{2.7}$$

$$\sum_{m=1}^{M} \sum_{t=ES_j}^{LS_j} (t + d_{jm}) F_{jmt} \leq \sum_{m=1}^{M} \sum_{t=ES_j}^{LS_j} tF_{jmt} \qquad \forall (i,j) \in PA \tag{2.8}$$

$$\sum_{j=0}^{n+1} \sum_{m=1}^{M_j} r_{jmk}^\rho \sum_{s=max\{t-d_{jm},ES_j\}}^{min\{t-1,LS_j\}} F_{jms} \leq R_k^\rho \qquad k = 1,\ldots,K^\rho \; ; \; t = 1,\ldots,T \tag{2.9}$$

$$\sum_{j=0}^{n+1} \sum_{m=1}^{M_j} r_{jmk}^\upsilon \sum_{ES_j}^{LS_j} F_{ims} \leq R_k^\upsilon \qquad k = 1,\ldots,K^\upsilon \tag{2.10}$$

$$x_{jmt} \in 0,1 \qquad j \in J; \; m \in M_j; \; ,t = ES_j,\ldots,LS_j \tag{2.11}$$

donde $ES_j(LS_j)$ es el tiempo más temprano (tardío) de inicio de la actividad j basado en los modos de menor duración.

La función objetivo (ecuación 2.6) minimiza la duración del proyecto. Se presupone que las actividades ficticias de inicio y fin solo se ejecutan en un único modo de duración cero y no consumen recursos.

El conjunto de restricciones 2.7 asegura que a cada actividad sólo se asigna un modo y un único tiempo de inicio. La ecuación 2.8 describe las relaciones de precedencia entre las actividades. Las restricciones representadas mediante la ecuación 2.9 aseguran que no se sobrepase la disponibilidad de los recursos renovables en cada periodo mientras que la ecuación 2.10 vela por los recursos no renovables a lo largo del proyecto. Finalmente, la ecuación 2.11 impone valores binarios a las variables de decisión.

Si en este problema solo se define un único modo de ejecución para cada actividad y no se especifican recursos no renovables, obtenemos el problema RCPSP presentado en la sección anterior.

Kolisch [55] demuestra que para el caso de dos recursos no renovables, el problema de asignación de modos es NP-completo y que el MRCPSP NP-duro.

En general, el software desarrollado para obtener la solución óptima del MRCPSP mediante el modelo de Talbot utiliza el método de fuerza bruta [36]. Sin embargo, tal como podría esperarse, los resultados obtenidos por dicho autor en pruebas preliminares son bastante malos. Por lo cual se hace necesario otro tipo de enfoque como métodos de ramificación y poda.

2.2.3. Generación de Programaciones Robustas

El problema que aquí se considera es la generación de secuencias base que sean estables ante las eventualidades que puedan presentarse durante la ejecución del proyecto, respetando las relaciones de precedencia y sin sobrepasar la disponibilidad de los recursos.

Suponemos un proyecto representado en el formato AON mediante un grafo dirigido $G = \{N, A\}$ donde N es el conjunto de nodos o actividades que deben ser procesadas y A es el conjunto de arcos que indican las relaciones de precedencia entre las actividades.

Las actividades no pueden interrumpirse y están numeradas desde 0 hasta $n+1$. Las actividades j=0 y j=n+1 son las actividades ficticias de inicio y fin de proyecto. La duración de las actividades se denota por $\mathbf{d}_j$. Las actividades no ficticias tienen una duración estocástica y las actividades ficticias tienen una duración de cero.

El tiempo de inicio de cada actividad se denota mediante S_j ($0 \leq j \leq n+1$) y el tiempo de finalización con F_j ($0 \leq j \leq n+1$). Existen K recursos renovables y cada actividad require para su ejecución una cantidad por periodo r_{jk}. La disponibilidad a_k del recurso k es constante a lo largo de la duración del proyecto.

Adicionalmente, cada actividad no ficticia tiene asociado un peso w_j que denota el costo marginal de comenzar en un instante de tiempo diferente al

planeado. El peso de la actividad ficticia fin w_n denota el costo de retrasar el proyecto más allá de una fecha determinada o fecha límite (δ_n).

Basándose en las definición de las variables y considerando que $P(t)$ representa al conjunto de las actividades que están siendo ejecutadas en el tiempo t, el problema puede formularse de la siguiente manera:

$$\text{minimizar} \sum_{j \in N} E|\mathbf{s}_j - s_j| \tag{2.12}$$

$$s_i + d_i \leq s_j \qquad \forall (i,j) \in A \tag{2.13}$$

$$\sum_{P(t)} r_{jk} \leq a_k \qquad \forall t, \forall k \tag{2.14}$$

$$s_{n+1} \leq \delta_n \tag{2.15}$$

La función objetivo presentada en la ecuación 2.12 permite maximizar la robustez de la solución minimizando la inestabilidad ponderada del proyecto. Se busca una programación que cumpla las restricciones de precedencia y de recursos y no exceda la fecha límite de finalización del proyecto y que no sea severamente afectada debido a los factores externos. Los tiempos de inicio realizados de las actividades son variables estocásticas. Las restricciones de la ecuación 2.13 hacen cumplir las relaciones de precedencia entre las actividades.

Las restricciones de la ecuación 2.14 aseguran que la demanda de cada recurso k no sea superior a su disponibilidad en cada periodo t. Finalmente, la ecuación 2.15 asegura que la finalización del proyecto (representada por el tiempo de inicio de la actividad ficticia fin) no exceda la fecha límite del proyecto. Se asume que no hay penalización por finalizar el proyecto antes de dicha fecha límite.

Leus & Heroelen [65] demostraron que el problema de programación de secuencias robustas pertenece a la clase de problemas NP-duro.

2.3. Métodos de Solución para el RCPSP y MRCPSP

Los problemas de Programación de Proyectos con Recursos Limitados RCPSP y MRCPSP, han sido estudiados en la literatura y existen numerosos trabajos que lo abordan desde diferentes perspectivas. Para resolverlos existen dos aproximaciones posibles: Métodos Exactos y Métodos Aproximados.

Los primeros aseguran la solución óptima al problema, aunque en general requieren un gran esfuerzo computacional, por tratarse de problemas NP-duro [11], [55] no logran resolver instancias de tamaño mediano y grande. Los segundos tratan de obtener buenas soluciones en un tiempo de cómputo razonable y son los que más auge y desarrollo han tenido en los últimos años. Por lo tanto, los esfuerzos en el desarrollo de nuevos métodos eficientes

están justificados. Las revisiones más recientes sobre métodos desarrollados se encuentran en los trabajos de Kolisch & Padman [58] y Demeulemeester & Herroelen [27].

Aunque la formulación es relativamente sencilla, requiere la utilización de un número elevado de variables y restricciones. En general, el software desarrollado para obtener la solución óptima del utiliza el método de fuerza bruta [36]. Sin embargo, tal como podría esperarse, los resultados obtenidos en pruebas preliminares son bastante malos.

Debido a ello, la formulación de este tipo de modelos hoy en día es más didáctico que práctico; sin embargo la relajación de algunas restricciones en los modelos propuestos por diversos autores ha permitido obtener nuevas cotas que han sido aplicadas con éxito en los algoritmos de enumeración implícita.

2.3.1. Métodos de Enumeración Implícita

Para asegurar que se encuentra una solución óptima al problema, el espacio de búsqueda debe ser examinado totalmente, resultando crucial descartar aquellas soluciones parciales que no conducen a soluciones óptimas. Por ello, la mayor parte de los métodos exactos para resolver este problema se basan en procedimientos de *Branch and Bound* (Ramificación y Poda)[27]. De forma general, estos procedimientos se basan en la enumeración de las diferentes soluciones posibles del problema a través del árbol de búsqueda.

La eficiencia de estos métodos depende de la cota inferior que se determine para el problema, la cual es obtenida por diferentes procedimientos, en particular relajando las restricciones del problema en cuanto a los recurso y luego resolviendo óptimamente el problema relajado. Cuanto más se relajen dichas restricciones más fácilmente se resuelve el problema relajado, pero peor es el valor obtenido de la cota. Adicionalmente se utilizan reglas de dominancia que permiten podar el árbol de búsqueda, eliminando aquellas ramas que no pueden conducir a la solución óptima. En general, la búsqueda es depth-first para mantener bajos requerimientos de memoria.

Los algoritmos de branch and bound para este problema se basan en el siguiente procedimiento: en cada etapa se va construyendo una secuencia parcial factible, en la cual están aquellas actividades a las que se les ha asignado un instante de comienzo (programadas). Cada vértice del árbol tiene asociado una secuencia parcial. El proceso de ramificación del árbol consiste en extender dicha secuencia, para lo cual existen diversos métodos como el basado en el *árbol de precedencias*, el basado en las *alternativas de retraso* y el basado ne las *alternativas de extensión*.

- Árbol de precedencias: Puesto que cada nodo del árbol de búsqueda representa una secuencia factible, en cada nodo se puede definir un conjunto de actividades elegibles, formado por aquellas actividades que aún no han sido programadas pero cuyas actividades predecesoras están dentro de la secuencia factible. Se escoge una actividad elegible y se programa tan

pronto como sea posible, expandiendo así una nueva rama del árbol. Por lo tanto, cada nodo del árbol tendrá tantas ramas como actividades elegibles haya en ese punto. El proceso termina cuando la actividad ficticia del fin del proyecto se convierte en elegible en un determinado nodo. problemas de tamaño pequeño y es una buena heurística para problemas grandes. Este método está basado en el trabajo de Talbot [92], fue repsentado por Patterson et al.[76] y mejorado por Sprecher [87].

- Árbol de alternativas de Retraso: Cada nodo tiene asociado un tiempo de secuenciación en el que pueden comenzar las actividades. En este caso, las actividades elegibles son aquellas que pueden comenzar en ese instante, entrando todas ellas al conjunto de actividades en proceso. Si la programación de estas actividades genera un conflicto de recursos, será necesario determinar los grupos de actividades en curso, cuyo retraso permite solucionar el conflicto originado. Estos grupos de actividades cuyo retraso resuelve el conflicto se les denomina alternativas de retraso. El instante de secuenciación viene determinado por el menor instante de entre los instantes de finalización de las actividades en proceso. Este algoritmo fue propuesto por Christofides et al. [20]. En el caso del MRCPSP se utilizan las así llamadas **Alternativas de modo y retraso** desarrolladas por Sprecher et al. [89] quienes extendieron el concepto de retraso de alternativas utilizado por Demeulemeester y Herroelen [26].
- Árbol de alternativas de extensión: Una alternativa de extensión es un conjunto de actividades elegibles que podría ser secuenciado en el instante actual sin exceder las disponibilidades de los recursos. Cada nodo del árbol origina tantas ramas como alternativas de extensión puedan formarse desde este nodo. Para la poda, se utiliza el criterio de primero el mejor, por lo que las necesidades de memoria para almacenar las soluciones intermedias son elevados. Para seleccionar el siguiente nodo para expandir la búsqueda utiliza un vector con distintas reglas de decisión e implementa varias reglas de dominancia y de poda, siendo la más eficiente la basada en la ruta crítica. A diferencia del método anterior, este no permite deshacer las decisiones tomadas previamente. Este procedimiento fue propuesto por Stinson [90]. Hartmann & Drexl [39] extienden esta técnica para el MRCPSP al incluir alternativas de modo y alternativas de extensión. Las alternativas de modo permiten fijar el modo de aquellas actividades elegibles a las que aún no se les ha asignado modo. En este procedimiento se combinan los conceptos de *Alternativa de modo* con *Alternativas de extensión*.

En el caso del RCPSP existe una aproximación completamente diferente basada en los denominados **conjuntos prohibidos mínimos**, concepto introducido por Igelmund & Radermarcher [45]. Un conjunto prohibido se define como un conjunto de actividades que pueden ser programadas concurrentemente durante el proyecto (es decir, que no existen relaciones de precedencia entre ellas) y que si se ejecutan de forma paralela violarían las

restricciones de recursos. El enfoque de conjuntos mínimos implica que se enumeran aquellos conjuntos prohibidos que no contienen dentro de si otro conjunto prohibido o, lo que es lo mismo, aquellos conjuntos prohibidos que al sustraerles una actividad dejarían de ser prohibidos.

En su procedimiento primero se determinan los conjuntos prohibidos mínimos, luego se procede a la destrucción de los mismos para obtener secuencias factibles de actividades. Esta destrucción se hace añadiendo tiempos de espera (que se modelan como relaciones de precedencia) entre al menos dos actividades de cada conjunto. Las diferentes relaciones de precedencias que se añaden dan lugar a las diferentes secuencias factibles que pueden formarse.

El enfoque desarrollado por Brucker et al.[17] define que para toda secuencia factible de actividades únicamente pueden existir tres tipos de relaciones entre cada par de actividades i - j: i precede a j, j precede a i o ambas pueden ejecutarse simultáneamente durante al menos una unidad de tiempo. Es obvio que entre algunos pares de actividades la relación existente entre las mismas queda fijada por las relaciones entre las actividades o por las necesidades de recursos. El proceso de búsqueda explora las posibles relaciones que pueden establecerse entre cada par de actividades no relacionadas.

Así, cada nivel del árbol de búsqueda representa los tipos de relación que pueden establecerse entre un par de actividades. Un camino en el árbol indica las relaciones elegidas para varios pares de actividades. Una secuencia se completa cuando se han seleccionado las relaciones para cada par de actividades. El proceso de enumeración del conjunto de secuencias factibles se acelera incluyendo algunos criterios de dominancia y la utilización de cotas inferiores y superiores.

De otra parte, Sprecher et al.[89] desarrollan un método de preproceso para el MRCPSP utilizado para reducir el espacio de búsqueda. Este preproceso permite excluir los modos no ejecutables e ineficientes de los datos del proyecto mediante un procedimiento que se ejecuta antes de cualquier método de resolución. La idea subyacente es excluir modos y/o recursos no renovables de los datos de entrada. Un modo es *no ejecutable* si su ejecución viola las restricciones de recursos renovables o no renovables. Un modo es *ineficiente* si existe otro modo para la actividad j que tanto su duración como su consumo de recursos no son menores que cualquier otro modo de la misma actividad. Finalmente, se eliminan los *recursos no renovables redundantes*, es decir aquellos en que la suma de los máximos requerimientos por cada actividad de dicho recurso no exceden su disponibilidad.

El preproceso se realiza en los siguientes pasos:

- Paso 1: Remover todos los modos no ejecutables de los datos del proyecto
- Paso 2: Eliminar todos los recursos no renovables redundantes
- Paso 3: Eliminar todos los modos ineficientes
- Paso 4: Si se ha eliminado algún modo en el paso tres, volver al paso 2

Este método es ampliamente usado en la solución del MRCPSP pro métodos exactos y aproximados. ([46], [37],[3], [104]).

A pesar de los esfuerzos realizados por desarrollar métodos robustos y rápidos, el tamaño del espacio de búsqueda crece desmesuradamente a medida que crece el número de actividades del problema. Esto se refleja en un crecimiento acelerado de los árboles de búsqueda y del tamaño y número de conjuntos prohibidos o posibles. Por ello se hace necesario la introducción de métodos que proporcionen soluciones factibles y de buena calidad en tiempos de cómputo razonable.

2.3.2. Métodos Aproximados

Tanto el RCPSP como el MRCPSP son problemas de la clase NP-duro no es posible resolver todas las instancias de manera óptima, pues los métodos exactos no pueden tratar con problemas de tamaño mediano o grande debido a la cantidad de variables y restricciones generadas y a la amplitud del espacio de búsqueda. La necesidad de obtener secuencias rápidas y de buena calidad solamente pueden ser obtenidas por medio del uso de métodos aproximados desarrollados para tal fin.

2.3.2.1. Métodos Basados en Reglas de Prioridad

Los métodos basados en reglas de prioridad son la base de la mayoría de métodos aproximados desarrollados para la programación de proyectos.

Este tipo de métodos parte de la programación obtenida mediante el método del camino crítico PERT/CPM, obteniendo distintos parámetros de las actividades del proyecto: fechas de comienzo y de finalización más tempranas y más tardías, holgura, etc. Estos datos se utilizan para construir diferentes reglas de prioridad que serán consideradas a la hora de decidir el orden en que las diferentes actividades van a ser programadas.

Un método basado en reglas de prioridad está conformado por dos elementos: un esquema generador de secuencias (SGS) y una regla de prioridad:

Esquema de Generador de Secuencias

El SGS determina la manera en que se construye una secuencia factible asignando tiempos de inicio a las distintas actividades. Se distinguen dos esquemas de secuenciación: serie y paralelo. Los dos esquemas generan secuencias factibles extendiendo por etapas una secuencia parcial. Una secuencia parcial es aquella donde solo un subconjunto de actividades tiene asignado un tiempo de finalización. En cada etapa el SGS construye el conjunto de todas las actividades elegibles, también llamado conjunto de decisión.

- Esquema Serie:Propuesto por Kelley (1963) y consiste en tantas etapas como actividades del proyecto. En cada etapa se selecciona una actividad de acuerdo con una regla de prioridad y se programa tan pronto como sea posible cumpliendo las restricciones de las relaciones de precedencia y sin sobrepasar la disponibilidad de los recursos. El algoritmo termina cuando todas las actividades están en el conjunto de actividades programadas. El conjunto de soluciones generadas por el esquema serie incluye siempre la solución óptima[53].
- Esquema Paralelo: Este método consiste en programar tantas actividades como sea posible en cada tiempo de decisión. El tiempo de decisión está asociado con la finalización de las actividades en ejecución.La aplicación del esquema paralelo pertenece al conjunto de las secuencias *sin retraso*, según ha sido probado por Kolisch [53], en las cuales no existe ningún periodo tal que una actividad que sea factible en cuanto a relaciones de precedencias y recursos en dicho periodo no haya sido programada. Aunque son secuencias en general más compactas, no siempre incluyen al menos una solución óptima.

Estos esquemas generadores de secuencias fueron extendidos para el caso de múltiples modos con recursos renovables por Lova & Tormos [67].

Los esquemas de generación de secuencias pueden aplicarse en dos direcciones: la programación hacia adelante, es decir cuando se comienza programando la actividad ficticia inicio y por último la actividad ficticia fin. Sin embargo, las actividades de una instancia de RCPSP pueden ser programadas en orden inverso, es decir se comienza con la actividad ficticia fin, programando gradualmente todas las actividades intermedias y se termina cuando se asigna tiempo de inicio a la actividad ficticia de inicio. Esto se puede hacer fácilmente invirtiendo todas las relaciones de precedencia, es decir los sucesores de la actividad se convierten en sus predecesores y los predecesores en sucesores.

Reglas de Prioridad

La regla de prioridad determina la actividad que se selecciona durante el proceso de búsqueda heurística. Estas reglas de prioridad generan una lista de actividades. En ella, las actividades quedan ordenadas según su prioridad, pero respetando las relaciones de precedencia, es decir, en una lista ninguna actividad puede estar antes que todas sus predecesoras.

Las reglas de prioridad son muy abundantes en la literatura, en el trabajo de Klein [49] se evalúan 73 de ellas y no existe ninguna que siempre tenga un desempeño mejor que las demás en todas las instancias. Demeulemeester[27] clasifica las reglas en cinco categorías según de donde proceda la información para construirla.

Reglas de prioridad para la selección de modo

En cada una de las etapas de los SGS cuando use selecciona una actividad, es necesario determinar el modo en que esta actividad será ejecutada para determinar su tiempo de duración y los recursos que utilizará. Boctor [12], [13] prueba tres reglas de prioridad para la selección de modos utilizando el esquema paralelo de secuenciación:

- P-SFM: el modo más corto posible en que puede realizarse una actividad. Un modo factible es aquel que puede ser ejecutado en el periodo actual dadas las disponibilidades de los recursos. Escoger el modo más corto parecería ser lo más adecuado para minimizar la duración del proyecto. Sin embargo, esta regla puede tener el efecto contrario. Usualmente los modos más cortos requieren una mayor cantidad de recursos y por tanto dificulta la ejecución de actividades de forma paralela.
- P-LCR: Menor razón crítica. Para un determinado recurso, la razón crítica es la relación de la utilización máxima del recurso (pico máximo) con la disponibilidad del mismo. Para ello se calcula el tiempo más temprano de las actividades del proyecto sin tener en cuenta la restricción de recursos. De allí se deriva el uso de recursos en cada periodo de ejecución. El recurso más crítico es aquel que tiene la mayor relación entre la utilización máxima del recurso (pico) y su disponibilidad. Se escoge el modo que requiera la menor cantidad del recurso más crítico. Con esta regla se trata de permitir que varias tareas se ejecuten de manera paralela.
- P-LRP: Menor proporción de recursos. Se escoge el modo que permite minimizar el criterio $max_r\{R_{m,r}/L_r\}$ donde $R_{m,r}$ denota la cantidad de recurso del tipo r que se necesita para la ejecución del modo m y L_r es la disponibilidad del recurso del tipo r.

Se hacen pruebas con problemas de 50 y 100 actividades, con 1,2,3 y 4 recursos renovables. De los resultados destaca que la regla SFM se desempeña mejor que las otras reglas de selección de modos. En Lova & Tormos [67] extienden el SFM para el esquema serie:

- S-SFM: para cada actividad selecciona el modo de ejecución tal que su tiempo de finalización sea lo más temprano posible (EFFT). De tal manera que todas las siguientes actividades pueden iniciarse lo antes posible. Si ocurre algún empate, el modo con mayor duración se selecciona.
  El uso del EFFT con el esquema paralelo de secuenciación produce los mismos resultados que la regla P-SFM.

Métodos Multipasada Basados en Reglas de Prioridad

El método de construcción de soluciones mediante regla de prioridades ha sido uno de los más utilizados debido a que es intuitivo y fácil de usar, es rápido en términos de esfuerzo computacional lo que permite que sea integrado con métodos de búsqueda local de inteligencia artificial.

Los métodos de una sola pasada (SGS+Regla de prioridad) pueden mejorarse fácilmente al incluir más de una pasada. Para ello se combinan los dos esquemas de secuenciación (serie y paralelo), las dos direcciones de secuenciación (backward y forward) y diferentes reglas de prioridad para obtener diferentes soluciones, la mejor de todas es la que se guarda como la solución de la instancia.

Durante algún tiempo los métodos multipasada fueron los que obtuvieron los mejores resultados en la resolución del problema.

Métodos Aleatorizados

Las heurísticas de una sola pasada y de pasadas múltiples son determinísticas en cuanto siempre obtienen siempre la misma secuencia final, aunque sean aplicadas varias veces. Una alternativa a este enfoque está en los métodos de sampling, en los cuales a cada actividad del conjunto de decisión se le asigna una probabilidad de ser seleccionada. Así en cada pasada se pueden obtener secuencias diferentes y la mejor es la que se reporta como programación final.

La probabilidad p_i con la cual la actividad i puede ser seleccionada puede ser determinada de distintas maneras. Dentro de los métodos aleatorizados, el más poderoso es el así llamado *Regret Based Biased Random Sampling* (RBRS) introducido por Drexl [28]. Este método calcula las probabilidades de manera indirecta utilizando el *regret value* ρ_i de cada actividad, el cual compara el valor de la prioridad de la actividad i con el valor más bajo entre todas las actividades del conjunto de decisión.

$$\rho_i = \begin{cases} \max\limits_{j \in DS} \nu(j) - \nu(i) & \text{si el objetivo es minimizar} \\ \nu(i) - \min\limits_{j \in DS} \nu(j) & \text{si el objetivo es maximizar} \end{cases}$$

El valor de la probabilidad se calcula mediante

$$\psi(i) = \frac{(\rho_i + \epsilon)^\alpha}{\sum_{j \in DS} (\rho_j + \epsilon)^\alpha}$$

Tal como está planteado hasta aquí, la actividad con valor más bajo en su prioridad no tiene ninguna probabilidad de ser seleccionada, por lo tanto si se adiciona la constante

$$\epsilon > 0$$

al *regret value* de cada actividad, entonces todas ellas tendrán alguna probabilidad de ser seleccionadas. Con esto se logra que el SGS pueda generar todas las secuencias posibles. Kolisch [52] reporta que el valor es $\epsilon = 1$.

Por medio del parámetro α, se puede controlar el sesgo del mecanismo. Si se escoge un valor de α lo suficientemente grande, el mecanismo no tendrá sesgo y habrá una selección de actividades determinística, basada en la regla de prioridad. Por el contrario, si $\alpha = 0$, el mecanismo será completamente aleatorio.

2.3.2.2. Métodos Metaheurísticos

El término *metaheurísticos* fue utilizado por primera vez por Glover[30] y su significado ha cambiado a lo largo de los años. En la actualidad, un algoritmo metaheurístico puede ser visto como una estrategia inteligente para diseñar o mejorar procedimientos heurísticos con un alto desempeño [70]. Por lo general combinan métodos constructivos, métodos de búsqueda local, conceptos que vienen de la Inteligencia Artificial, Métodos Estadísticos y Métodos Bioinspirados.

A continuación se presentan los métodos y los trabajos más relevantes en la solución del RCPSP.

Tabu Search

El método Tabu Search (TS) o Búsqueda Tabú fue introducido por Fred Glover[30] y su principal característica es el uso de memoria adaptativa que permite explorar diferentes regiones en el espacio de búsqueda (memora de corto plazo) e intensificar la búsqueda en áreas promisorias (memoria de largo plazo). El libro de Glover y Laguna [31] presenta una revisión del tema.

Baar[9] desarrolla dos versiones del TS para el RCPSP. La primera está basada en una lista de actividades y el esquema serie como procedimiento de decodificación. El vecindario se genera utilizando una de las tres clases de movimientos basados en la ruta crítica. La segunda versión está basado en la así llamada representación de esquema de secuencia y utiliza el mecanismo de generación de vecinos y una regla de decodificación presentada por Brucker [17].

El método desarrollado por Thomas & Salhi [93] opera directamente en las secuencias definiendo tres movimientos diferentes. Este enfoque necesita un procedimiento para asegurar que todas las secuencias generadas son factibles. Klein [49] propone un procedimiento llamado RETAPS (Reactive Tabu Search for Project Scheduling) y fue probado en las instancias de 30 y 60 trabajos de la librería PSPLIB[59] con muy buenos resultados.

Nonobe e Ibaraki [74] proponen una búsqueda tabú utilizando una lista de actividades, y un mecanismo de reducción del vecindario. En su enfoque permiten también el caso de modos múltiples y la utilización de recursos no renovables. El algoritmo es probado con los tres conjuntos de instancia de PSPLIB, dando resultados muy cercanos al algoritmo de Klein.

El uso de los algoritmos de búsqueda tabú en el RCPSP ha dado resultados interesantes, los que obtienen mejores resultados en la librería PSPLIB utilizan la representación de lista de actividades y el esquema serie para decodificar la solución.

Simulated Annealing

Este método fue desarrollado por Kirkpatrick et al.[48] y está basado en el enfriamiento de materiales.

El algoritmo de Boctor[14] resuelve el RCPSP y lo compara con el TS de Pinson[78] y concluye que es más eficiente el primero. Adicionalmente el operador de cambio que se desarrolla en este trabajo se utiliza como mecanismo de mutación en Algoritmos Genéticos.

Slowinski et al.[86] proponen un sistema de apoyo a la decisión (DSS) para un MRCPSP multiobjetivo que ayuda al usuario a identificar estrategias para escoger las actividades que deben comenzarse en caso de que exista un conflicto de recursos o haya múltiples objetivos. El DSS contiene tres estrategias diferentes de solución: heurísticas de una sola pasada, heurísticas de múltiples pasadas y el uso de la metaheurística de recocido simulado. El núcleo de todas las estrategias de solución es una lista de actividades factible en cuanto a precedencias derivada de una de 12 prioridades.

El DSS principalmente se propone como un prototipo y no hay un riguroso estudio computacional.

Jozefowska et al.[46] desarrollan dos versiones para resolver el MRCPSP. La primera tiene en cuenta únicamente las soluciones factibles con respecto a las relaciones de precedencia y a los recursos, mientras que en la segunda también se incluyen las soluciones no factibles con respecto a la utilización de recursos no renovables. Utilizan el esquema de preprocesamiento de Sprecher [89] y las soluciones se representan mediante dos listas: una lista de actividades y una lista de asignación modos y utilizan el esquema serie como procedimiento de decodificación. En la versión que maneja soluciones no factibles el valor objetivo es a función objetivo presentada en Hartmann [37].

Para el estudio computacional se utilizan los conjuntos de la librería de pruebas PSPLIB. Los resultados muestran que el algoritmo que admite soluciones no factibles obtiene mejores resultados.

El procedimiento de Bouleimen y Lecocq [15] cubre tanto el RCPSP como el problema de modos múltiples. En el MRCPSP utilizan dos listas de representación: una lista de modos y una lista de actividades. Utilizan el esquema serie como método de decodificación para generar la secuencia del proyecto. El algoritmo tiene dos fases: en la primera fase utilizan un operador de cambio de modo para encontrar una asignación de modos factible y en la segunda fase utilizan el operador de inserción de Boctor para generar nuevas listas de actividades para dicha asignación de modos. El método desarrollado es eficiente al ser probado en las instancias de prueba de Kolisch (PSPLIB).

Finalmente, Valls et al.[99] diseña un simulated annealing y utiliza los métodos de mejora forward backward, siendo también eficiente en la librería de pruebas.

Algoritmos Genéticos

Los Algoritmos Genéticos (AG) fueron introducidos por John Holland [1975] y están inspirados en la evolución de los seres vivos. La idea básica es que durante la evolución, los individuos mejor adaptados tendrán más probabilidades de sobrevivir y reproducirse, mientras que los menos adaptados

son eliminados [32]. Cada individuo de la población es una solución al problema cuya solución está codificada en la información genética. Cada individuo tiene asociado un valor de su aptitud, que muestra la calidad de la solución.

Los principales componentes son la codificación de los n individuos, las funciones de evaluación, selección, cruce y mutación. La población inicial (p) está conformada por soluciones factibles que pueden ser generadas aleatoriamente. La función de evaluación asigna un valor (*fitness*) a cada individuo y mide la calidad de dicha solución.

Una vez evaluada, la función de selección determina cuales individuos generarán a los nuevos. Existen diferentes tipos de selección utilizan directa o indirectamente los valores de fitness para guiar el proceso de búsqueda de nuevas y mejores soluciones.

Los operadores tradicionales de cruce intercambian información entre los padres, de manera probabilística determinan en cada bit cuál de los padres aportará la información para construir la nueva solución. Los operadores de mutación modifican la información del individuo para introducir diversidad a la población.

Una vez todos los nuevos individuos están construidos (nueva generación) reemplazan a la población anterior. Para evitar la pérdida de la mejor solución, esta es trasladada directamente de la población anterior a la nueva (elitismo).

Como esta tesis aborda la solución de la programación de proyectos con recursos limitados mediante el uso de algoritmos genéticos, se profundizará ampliamente sobre el funcionamiento de los AG y de cada uno de sus operadores en la sección 2.5.

Los AG han sido ampliamente utilizados para resolver problemas de programación de proyectos. Lancaster & Ozbayrak [63] hacen una revisión sobre el tema. En cuanto al RCPSP los AG son el método más utilizado para su solución. Destacan los trabajos desarrollados por Alcaraz y Maroto[1], [2]; Debels [24]; Hartman [38]; Valls et al.[99],[98].

Su uso para resolver el caso multimodo también es extendido en la literatura. En el trabajo de Wall [105] sólo tiene en cuenta recursos renovables periodo a periodo. Los trabajos de Mori y Tsen [73], Hartmann [37], Alcaraz et al.[3] y Van Peteghem y Vanoucke [104] abordan el problema completo. Muchos de ellos son la extensión de métodos propuestos para el RCPSP incluyendo el aspecto de múltiples modos.

2.3.3. Métodos de Mejora de Programaciones Factibles

Los métodos de mejora de programaciones factibles están basadas en el concepto de holgura libre backward/forward de una actividad. La holgura libre es el tiempo que una actividad tiene para ser atrasada/adelantada sin que las otras actividades se vean afectadas en sus tiempos de finalización o inicio, respectivamente.

2.3.3.1. Método de Mejora Forward-Backward para el RCPSP

El método de mejora Forward-Backward (FBI) desarrollado por Tormos y Lova [94] está basado en las listas de actividades y el esquema serie para mejorar programaciones factibles obtenidos por cualquiera de los métodos descritos anteriormente. También ha sido denominado Técnica de Doble Justificación [99].

En esta técnica, consiste en hacer al menos una pasada forward y una backward a una secuencia factible. En la **pasada backward** las actividades son reordenadas en orden decreciente de su tiempo de finalización y se aplica el esquema serie a esta nueva secuencia. Como resultado las actividades se mueven a la derecha a la posición más tardía posible.

En la **pasada forward** las actividades se mueven al momento más temprano posible que permita su holgura libre. Para ello las actividades de la secuencia factible son ordenadas en orden creciente de su tiempo de inicio y se aplica el esquema de generación serie a esta nueva secuencia.

Tal como fue presentado en el trabajo original [94], el método consiste en tomar una secuencia factible e iterativamente hacer pasadas backward/forward (BF) haste que en dos pasadas BF consecutivas no hubiera mejora en el tiempo de duración del proyecto.

2.3.3.2. Método Paralelo de Mejora para MRCPSP

Este procedimiento es presentado por Özdamar [75] y se aplica luego de que el método paralelo ha decodificado. Se actualizan las ventanas de tiempo para cada actividad: primero se calcula el tiempo más tardío de finalización de la actividad, teniendo en cuenta la duración total del proyecto al hacer la secuencia con el método paralelo forward: $LFT_i = C_{max} - ST_i$ y luego el tiempo más temprano de inicio $EST_i = C_{max} - FT_i$. Con estas ventanas de tiempo se actualizan las prioridades de las actividades y se vuelve a generar la secuencia utilizando la misma información en cuanto a reglas de prioridad, empezando por la última y terminando por la primera.

2.3.3.3. Método Serie de Mejora para MRCPSP

Este procedimiento es presentado por VanPetgem y Vanhoucke [104]. Una vez generadas las secuencias, éste método de decodificación se aplica a la secuencia con una probabilidad p_{modopt} determinada. El procedimiento decodifica las actividades según su prioridad. Para cada actividad analiza si es posible ejecutarla en un modo tal que permita que finalice antes de su tiempo actual de finalización sin demejorar el valor de la capacidad sobrante de recursos $L^{\nu}(I)$.

2.3.3.4. Justificación a la izquierda versión multimodo

Presentado por Hartmann[37], se basa en la definición de la justificación izquierda versión multi-modo que fue originalmente definido como regla de acotamiento del algoritmo exacto de Sprecher et al.[89].

La justificación izquierda versión multimodo de una actividad j en una programación dada es una operación que reduce el tiempo de finalización de la actividad sin cambiar los modos o el tiempo de finalización de las otras actividades y la secuencia debe sin violar las limitaciones. Por lo tanto, la actividad j puede cambiar de modo.

Este método tiene dos características importantes: considera los modos y los tiempos de inicio simultáneamente y segundo, nunca se empeora la duración del proyecto de la secuencia actual. Puede ser de una única pasada o aplicarse iterativamente en múltiples pasadas.

- **Mejora de una pasada**: En esta versión se comienza por la actividad 1 hasta la n y se verifica si se puede hacer un movimiento a la izquierda. Para ello se comprueba si la actividad puede ser ejecutada en un modo de menor duración sin violar las restricciones de capacidad. Se comienza por el modo 1 (es el de mas corta duración) y se selecciona el primer modo que permite el cambio y se actualizan las disponibilidades. Se salta la siguiente actividad en la lista y se sigue el proceso con la actividad siguiente. Se le llama de una sola pasada porque cada actividad es considerada una única vez para la justificación a la izquierda.
- **Mejora de múltiples pasadas**: Como en la mejora de una sola pasada cada actividad es considerada una única vez, la secuencia generada no es necesariamente la más ajustada a la izquierda que se pueda conseguir. Una secuencia completamente justificada a la izquierda es aquella en la que ninguna actividad puede ser comenzada en un instante previo al actual. Por tanto parece lógico ejecutar el proceso anterior tantas veces como se haga necesario hasta que no sea posible ejecutar ningún cambio a la izquierda. Este proceso iterativo es el método de mejora de múltiples pasadas.

El último paso de las mejoras utilizadas por Hartmann consiste en rehacer la lista de actividades del Individuo que ha sido mejorado. Para ello las actividades en la secuencia mejorada se ordenan en orden creciente con respecto a sus tiempo de inicio y se pasan a la lista de actividades. El modo correspondiente a la actividad se guarda en la lista de modos.

La experiencia computacional[37] demostró que el método de mejora de una sola pasada y sin herencia obtenía mejores resultados independientemente del tamaño de la población o un algoritmo genético sin mejora. Hace también un análisis de similitud de la población, y determina que al usar la herencia el número de individuos similares (clusters) desciende rápidamente de tal manera que al cabo de 50 generaciones (cada generación con 60 individuos) solo hay dos clusters diferentes.

2.4. Generación de Programaciones Robustas

A pesar de que el entorno habitual de la mayoría de los estudios de la programación de proyectos es aquel donde se asumen condiciones determinísticas de trabajo, en la mayoría de entornos prácticos la programación "off-line" tiene una aplicación limitada debido a la variación de las condiciones de ejecución.

En este problema pueden distinguirse tres tipos de secuencias, necesarias todas para la generación y evaluación de soluciones robustas:

- **Secuencia inicial**: Como se verá más adelante, autores como Herroelen [43] o Van de Vonder et al.[100], [101] utilizan un esquema de dos fases, en la cual la construcción de la secuencia base parte de una secuencia inicial no protegida (sin buffers). Esta secuencia puede ser generada por cualquier procedimiento que se considere adecuado.
- **Secuencia base**: o secuencia predictiva es una lista del tiempo de inicio de las actividades del proyecto. Esta secuencia se construye asumiendo un entorno estático y determinístico y se calcula antes de que comience la ejecución del proyecto.
- **Secuencia realizada**: La secuencia realizada es la lista del tiempo de inicio de las actividades una vez se ha ejecutado el proyecto. El método proactivo y las decisiones reactivas que se toman a medida que se ejecuta el proyecto tienen una gran influencia en la secuencia realizada.

La secuencia base cumple múltiples funciones [8]. Primero, provee visibilidad dentro de la organización de las ventanas temporales que deben reservarse para la ejecución de las actividades para reflejar los requerimientos de personal, equipo y otros recursos. La secuencia base es el punto de partida de la comunicación y coordinación con entidades externas a la organización y constituye la base para acuerdos con los proveedores y subcontratistas, así como los acuerdos con los clientes en cuanto a fechas de entrega. Una decisión importante en la construcción de la secuencia base es la cantidad de tiempo de seguridad que se incluirá en ella.

Una posibilidad para maximizar la robustez de la solución es incluir tiempos de seguridad en la secuencia base de tal manera que sea posible absorber las disrupciones que pueden ser previstas. Este enfoque se llama *programación proactiva*. Un segundo enfoque, consiste en definir un procedimiento que reaccione ante las disrupciones que no pueden ser absorbidas por la secuencia base. Este enfoque se llama *programación reactiva*.

Un enfoque puramente proactivo sería irrealizable. La incorporación de tiempos de seguridad en la secuencia base de tal manera que se puedan absorber todas las disrupciones posibles llevaría a construir una secuencia base con un tiempo de duración demasiado largo. Todos los métodos de construcción proactiva de secuencias base deben estar acompañados de un método reactivo.

Sin embargo, el primer paso es definir de manera concreta lo que se entiende por robustez y la manera de medirla.

2.4.1. Tipos de robustez y medidas

Previamente hemos introducido el concepto de robustez de la secuencia como la estabilidad que ésta presenta a las posibles disrupciones que pueden ocurrir durante la ejecución del proyecto. Herroelen & Leus [44] distinguen dos clases de robustez:

- *Robustez en la calidad* que hace referencia a la protección de la medida de desempeño que se use para evaluar la calidad de la secuencia; por ejemplo que el proyecto no exceda su fecha límite de finalización.
  Para medir la robustez en la calidad puede medirse con respecto al valor esperado de la función objetivo como lo hace Stork para el caso estocástico [91]. Herroelen & Leus [41] calculan el porcentaje de desviación de la duración con respecto a un valor óptimo que se calcula una vez finalizado el proyecto. Finalmente, Van de Vonder et al.[102] definen el nivel de servicio como la probabilidad de que la medida de desempeño de la secuencia realizada permanezca dentro de un cierto límite.
- *Robustez de la solución* que hace referencia a la estabilidad de la secuencia (tiempo de inicio de las actividades). Esto significa que dada la incertidumbre durante la ejecución se quiere que la secuencia ejecutada difiera lo menos posible de la secuencia base.
  Para medir la robustez de la solución la mayoría de los autores utilizan datos relacionados con la holgura de las actividades debido a que si una actividad tiene holgura puede modificar su tiempo de inicio o fin sin afectar otras actividades en la secuencia. Cesta et al.[19], Schwindt [85], Herroelen & Leus [42], Lambrechts et al.[62] utilizan medidas relacionadas con la holgura libre de las actividades como medida surrogada de la robustez de la solución. Otro enfoque es el de Policella[80] o Aloulou & Portmann [5] que calculan la flexibilidad de la solución por medio de la cantidad de actividades no relacionadas entre si.

Sin embargo el desempeño de la medida de robustez se ve afectado tanto por los métodos proactivos de construcción de secuencias base, como por las políticas reactivas para resolver las infactibilidades que puedan presentarse durante la ejecución. En las próximas secciones se hace una revisión de estos métodos

2.4.2. Métodos Proactivos

El objetivo de los métodos proactivos es la construcción de secuencias base robustas, es decir secuencias que estén tan protegidas como sea posible frente a las eventualidades que puedan surgir durante la ejecución del proyecto. Se

da por supuesto que es posible que una secuencia robusta, a pesar de estar protegida, se convierta en infactible durante la ejecución del proyecto.

La manera de proteger las secuencias base es mediante la inclusión de tiempos de seguridad (buffers) entre actividades. Por lo general se considera una fecha determinada de finalización del proyecto (δ_n) y una secuencia inicial sobre la cual se insertan los buffers par protegerla de eventos inesperados.

Cualquier método de solución para el RCPSP usando la duración media de las actividades sirve para construir la secuencia inicial sin buffers. Los buffers son tiempos muertos en la secuencia que se inserta entre las actividades. Éstos actúan como un colchón para prevenir la propagación de los retrasos primarios a través de toda la secuencia. Los métodos que se describen a continuación son los más relevantes en la literatura de programación de proyectos.

1. Método de la cadena crítica: El método del cadena crítica (CC/BM), desarrollado por Goldratt[34] es la aplicación directa de la Teoría de las Restricciones (TOC) en el área de la gestión de proyectos. Ha recibido una atención considerable en la literatura de la administración de proyectos. Su nombre deriva del concepto de *cadena crítica* que es la secuencia de actividades más larga considerando tanto dependencias de tareas como de recursos, y que por tanto determina la duración del proyecto.
   Su principal característica en cuanto a la planificación de un proyecto es que la tareas individualmente deben ser despojadas de su protección para concentrarla en determinados puntos con el fin de dotarle de una mayor fortaleza a la hora de hacer frente a posibles desviaciones, siendo así necesaria una menor protección global.
2. Heurística RDFF: Propuesta por Van de Vonder et al.[103]. En esta heurística los tiempos de seguridad, a modo de buffers, antes de aquellas actividades que pueden generar los costos más altos si comienzan más tarde de lo planeado. Para calcular el posible costo asociado a cada actividad se hace una red de flujo de recursos siguiendo el procedimiento de Artigues & Roubellat [7].
3. Heurística VADE: Esta heurística propuesta por Van de Vonder et al.[100] basa el impacto de la actividad j en la desviación estándar de su duración σ_j, la cual se asume conocida. Con base en la desviación estándasr de la duración de cada actividad no ficticia del proyecto se calcula de forma iterativa una extensión virtual de su duración. La duración virtual se utiliza para actualizar el tiempo de inicio de las actividades y por tanto de la inserción de buffers en la secuencia inicial. Los tiempos de inicio actualizados se utilizan para generar la secuencia base utilizando la duración original de las actividades.
   La selección de la actividad a la que se extenderá su duración tiene en cuenta tanto la variabilidad de la actividad como el número de iteraciones hecha por el método. Se continua con la extensión virtual de la duración mientras que la duración del proyecto sea menor al plazo de finalización fijado.

4. Heurística STC: Propuesta por Van de Vonder el al.[100] y está basada en la criticidad de los tiempos de inicio de la actividad (STC). La idea básica es empezar desde una secuencia inicial sin ningún buffer e iterativamente crear secuencias intermedias insertando un buffer de una unidad de tiempo antes de la actividad cuyo tiempo de inicio sea el más crítico en la secuencia intermedia actual, hasta que la adición de más tiempo de seguridad no implique una mayor estabilidad.
5. Búsqueda Tabú: Van de Vonder et al.[100] proponen el diseño de una búsqueda tabú que permita explorar una zona más amplia de la región factible. Para cada actividad no ficticia existen dos tipos de vecindarios. Uno se obtiene al incrementar el buffer que está antes de la actividad en un periodo de tiempo (*movimiento mas*). El otro se obtiene al decrementar el tamaño del buffer en una unidad *movimiento menos*. Los buffers de las otras actividades se mantienen sin cambios. Se mantienen dos listas tabú, ambas con una longitud de $\lfloor n/3 \rfloor$. Cada una almacena un tipo de movimiento. En cada iteración, el vecindario contiene $2\times(n-2)$ soluciones.

2.5. Algoritmos Genéticos

Los Algoritmos Genéticos son uno de los métodos aproximados que mejores resultados proporcionan al resolver los problemas de programación de proyectos con recursos limitados en problemas que no pueden ser resueltos por los métodos exactos. En esta sección se hace una revisión detallada de los diferentes aspectos que involucra el diseño y programación de un AG y los diferentes trabajos que los utilizan.

2.5.1. Representación de las Soluciones

El AG opera sobre una representación codificada de las soluciones, equivalente al material genético de un individuo, y no sobre las soluciones como tal. Una adecuada codificación de las soluciones es crucial para el éxito del algoritmo y es la base de la que dependen el resto de operadores, que deben ser diseñados para que utilicen de manera eficiente las propiedades de cada representación.

Kolisch [56] distingue 5 tipos de representación de las soluciones. La más utilizada es la **Lista de Actividades**, también llamada "codificación basada en permutaciones". Esta representación es una lista donde las actividades son ordenadas según su prioridad, siendo la primera la más prioritaria y la última la menos prioritaria. Adicionalmente, ninguna actividad puede incluirse antes de que en ella estén todos sus predecesores; manteniendo así la factibilidad con respecto a las relaciones de precedencia.

Esta lista es decodificada utilizando un esquema de generación y una dirección de planificación. Se obtiene, así, una programación a partir de la lista. Es importante destacar que una programación determinada puede ser

representada por diferentes listas. Más recientemente, se han incluido genes adicionales para representar el tipo de esquema de generación de secuencia (S/P) y la dirección de programación (B/F).

Según los resultados experimentales la representación basada en la lista de prioridades es la más adecuada para el RCPSP[35], [1]. La representación por lista de actividades ha sido utilizada por Hartman [35], [38]; Alcaraz & Maroto [1], [2]; Debels & Vanhoucke [24]; Valls et al.[99] y [98].

En el caso de MRCPSP cada individuo tiene una lista adicional correspondiente al modo de ejecución de la actividad. Esta representación asume que el plan existe, los modos de ejecución deben estar completamente definidos y las restricciones y objetivos deben ser definidos *a priori*. Alcaraz et al.[3] extienden esta representación al incluir un gen adicional para la dirección en que será decodificada la lista de actividades y de modos.

Para generar la población inicial en el MRCPSP Hartmann [37] utiliza el siguiente procedimiento: Se asigna el modo de ejecución para todas las actividades y se verifica el cumplimiento de la restricción de capacidad de los recursos no renovables. Si no se satisface se selecciona aleatoriamente una actividad y se cambia su modo. Si este modo es al menos tan bueno como el anterior (que la capacidad sobrante sea igual o menor al modo anterior).

Este proceso se repite hasta que la solución sea factible o hasta un número máximo de intentos consecutivos sin mejorar la solución. Una vez fijados los modos, se construye la lista de actividades usando MinLFT derivado de los tiempos de duración dados por los modos de ejecución seleccionados.

2.5.2. Tamaño de la Población

El tamaño de la población es una de las decisiones más importantes que se deben tomar al diseñar un AG. Si la población es muy pequeña, el AG puede converger muy rápidamente; y si es muy grande el AG desperdicia recursos computacionales al gastar mucho tiempo en la mejora de los individuos.

- Población Fija: Tradicionalmente los AG tienen un tamaño de población fija cuyo valor en muchos casos atiende a la "*configuración estándar*" de 50 o 100 individuos en la población. También puede estar determinado por los parámetros del problema que se resuelve y se realizan algunos experimentos con diferentes tamaños de población para determinar cuál se comporta mejor. En el caso del RCPSP el tamaño de población se asocia con el número de actividades del proyecto.
- Población Variable: Debido a que el tamaño de la población es un factor crítico, se han propuesto diversos métodos para fijar el tamaño de la población. Lobo y Lima [66] hacen una revisión de los diferentes métodos de ajuste de la población que puede hacer un AG de manera autónoma.

En la actualidad los AG diseñados para la solución del RCPSP o MRCPSP utilizan un tamaño de población fijo. En particular, en el RCPSP muchos

autores fijan el tamaño de la población según el número de soluciones que se vayan a generar. Entre más soluciones, más grande es la población.

2.5.3. Selección

Una parte fundamental del funcionamiento de un AG es el proceso de selección de los candidatos a generar los nuevos individuos de la población. Se debe garantizar que los mejores individuos tengan una mayor posibilidad de ser padres (reproducirse) frente a los individuos menos buenos. Mientras más apto sea un organismo (valor de aptitud más alto), más veces será seleccionado para reproducirse.Sin embargo, los individuos menos adaptados también deben tener probabilidades de reproducirse, debido a que su material genético puede ser útil para encontrar buenas soluciones.

La **presión selectiva** afecta el proceso de búsqueda. Si es muy fuerte los organismos de alta adaptación que son subóptimos pueden dominar la población, disminuyendo diversidad necesaria para progresar. Por el contrario, si es muy débil la evolución y convergencia son lentas.

En general, la selección se realiza de forma probabilística, es decir que aunque un individuo esté poco adaptado tiene probabilidad (baja) de reproducirse. De los diversos métodos de selección propuestos en la literatura, los siguientes son los más utilizados en el contexto de la programación de proyectos:

1. Ruleta: El método de la ruleta ha sido el método más comúnmente utilizado desde el origen de los AG. Los padres se seleccionan de acuerdo con su aptitud. Los individuos mejores son los que tienen mayores posibilidades de ser elegidos, siendo la probabilidad de ser seleccionados proporcional a su aptitud. Este método es utilizado por Hartmann [35].
2. Torneo: La idea básica de este método es seleccionar con base en comparaciones directas de los individuos. En la manera más común de implementarlo se comparan p individuos de la población escogidos al azar y se selecciona el que tenga mejor aptitud. El tamaño del torneo es p, y usualmente $p = 2$. Cuando $p \geq 10$ la selección se considera dura y se considera blanda cuando $2 \leq p \leq 5$. Este tipo de selección ha sido utilizada por Hartman [35], Alcaraz y Maroto [1], [2]. En Debels y Vanhoucke [24] un padre es escogido por torneo y el otro es el $i - esimo$ individuo de la población. En cuanto al MRCPSP Van Petghem[104] utiliza el método de 2-torneo para seleccionar cada uno de los padres a los que se les aplicarán los operadores genéticos.
3. Ranking: Este método fue diseñado para subsanar el tema de la convergencia prematura que puede surgir cuando existan grandes diferencias entre los fitness de los individuos de la población. Por ejemplo si un cromosoma ocupa el 90 % de la ruleta el resto de los cromosomas tienen muy pocas posibilidades de ser elegidos. La selección por ranking da solución a este problema, los individuos son ordenados de acuerdo a su

ranking de fitness. De esta manera si tenemos n cromosomas el individuo con peor fitness se le asignará un 1 y el que tenga el mejor fitness se le asignará la n. Una vez más, en el caso de minimizar (como en el RCPSP) se debe transformar la aptitud de los individuos. Este método es utilizado Alcaraz y Maroto [1] para resolver el RCPSP y por Hartman para resolver el RCPSP [35], [38] y el MRCPSP [37].

2.5.4. Cruce

En los sistemas biológicos, el cruce es un proceso complejo entre parejas de cromosomas. Estos cromosomas se alinean, luego se fraccionan en ciertas partes y posteriormente intercambian fragmentos entre sí.

Este operador permite el intercambio de información entre las secuencias de la población actual, por lo tanto tiene un gran impacto en el resultado del algoritmo. Es por ello que es uno de los operadores genéticos que más atención ha recibido por parte de los investigadores. A continuación describimos los que se emplean con más frecuencia en la programación de proyectos.

1. Cruce de Un Punto: Esta técnica fue propuesta por Davis [22]. En un cromosoma de longitud n existen $n-1$ puntos de cruce, por lo que el punto de cruce k será un número aleatorio en el intervalo [0, $n-1$]. El hijo tendrá los genes del 0 al k iguales a los de su padre y los genes desde $k+1$ hasta n heredados en el orden relativo de su madre. Este procedimiento asegura que la secuencia generada sea factible. La hija se manera de manera análoga.
   Esta técnica ha sido utilizada por Hartmann [35]. Una versión modificada de esta técnica es empleada por Alcaraz & Maroto [1], quienes utilizan el gen que representa la SD para generar el hijo de inicio a fin si el gen es Forward o de fin a inicio si el gen es Backward.
   Este operador fue extendido por Hartman[37] para la versión multimodo y también es usado en el trabajo de Van Petghem [104].
2. Cruce de Dos Puntos: Esta técnica fue propuesta por Goldbert & Lingle [33] y puede verse como una generalización del operador anterior. Para implementar este operador, se generan dos puntos de cruce de manera aleatoria k_1 y k_2, y $k_2 > k_1$. El modo más común de realizar este cruce consiste en que el hijo hereda directamente del padre los genes del intervalo inicial $[0, k_1]$ y del intervalo final $(k_2, n]$. Los genes del intervalo $(k_1, k_2]$ se heredan en el orden relativo de la madre. La hija se genera de manera análoga.
   Este operador es utilizado por Debels & Vanhoucke [24], Hartman [35], Alcaraz & Maroto [1], [2]. Adicionalmente han modificado el operador para incluir la información codificada en el gen de la dirección de programación: si el gen es Forward el hijo se genera de inicio a fin y si es Backward se genera de fin a inicio.
   En el caso de múltiples modos este operador es utilizado por Özdamar [75] y por Alcaraz et al.[3].

3. Cruce Uniforme: Esta técnica se trata de un cruce de n puntos, en el cual el número n no se fija previamente. Se genera una lista de números aleatorios $p_i \in \{0, 1\}$, $i = 1, \ldots n$. De manera sucesiva se llenan las posiciones $i = 1, \ldots n$ en el hijo. Si $p_i = 1$ se toma la actividad de la lista del padre que tenga el menor índice entre las actividades no seleccionadas y si $p_i = 0$ la actividad se toma de la lista del padre. La hija se genera de manera análoga heredando de la madre si $p_i = 1$ y del padre si $p_i = 0$.
   Este tipo de cruce es utilizado por Hartamann [35] y Mendes et al.[71] y en el MRCPSP es utilizado por Wall [105] para la lista de modos.
4. Peak Crossover: Este operador fue diseñado específicamente para el RCPSP por Valls et al.[97] y permite combinar partes específicas de la solución que contribuyen con la calidad del individuo. En el caso del RCPSP es deseable que los hijos hereden los periodos donde se utilizan intensivamente los recursos. Este operador selecciona las secuencias parciales donde el uso de recursos es superior a un límite determinado y las hereda directamente al hijo.
5. Blend Crossover: Utilizado por Wall [105] en la solución del MRCPSP y fue diseñado específicamente para el arreglo que contiene los tiempos entre actividades. Este operador genera un nuevo valor dependiendo de la distancia (similitud) de los padres. Es un operador adaptativo y no tiene necesidad de ajustar ningún parámetro. Si p_1 es el valor del gen del padre y p_2 el de la madre y suponemos que $p_1 > p_2$, se calcula la distancia entre los padres como $I = |p_1 - p_2|$ y $\alpha = d/2$. El valor del hijo es un valor aleatorio en el intervalo $[p_1 - \alpha I, p_2 + \alpha I$. Los valores se truncan según los datos del problema para mantener la factibilidad de los tiempos de inicio de las actividades.

2.5.5. Mutación

La mutación se considera como un operador secundario en los AG. Algunos investigadores sugieren usar porcentajes altos de mutación al comienzo de la búsqueda y luego decrementarlos exponencialmente. Otros autores sugieren que $p_m = \frac{1}{l}$, donde $l \leq L$ y L es la longitud de la cadena cromosómica. Sin embargo lo más frecuente es que el valor se fije al inicio del algoritmo y se usen probabilidades de mutación inferiores al 5 %. A continuación se revisan los principales operadores de mutación.

1. Flip Bit: Este es el operador más sencillo de mutación y es utilizado únicamente cuando cada gen puede tomar solo dos valores, por ejemplo 0 y 1. Este operador cambia el valor de 0 a 1 y de 1 a 0 cuando es utilizado. Se implenta cuando la representación de los individuos incluye un gen para el tipo de codificación (Serie o Paralelo) y un gen para la dirección de programación (Forward o Backward).
2. Inserción: Desarrollado por Boctor[14], es el operador que mejores resultados tiene para este problema, pues siempre crea secuencias factibles.

Si se quiere aplicar este operador a una actividad j_i que está en una lista de actividades $\lambda = \{j_1, \ldots, j_i, \ldots, j_h, \ldots, j_n\}$, debe buscarse la posición en la lista de todos sus predecesores y se guarda la máxima (maxpred) y la posición que ocupan sus sucesores y guardar la mínima (minsuc). Luego, se genera un número aleatorio en dicho intervalo y en dicha posición se inserta la actividad. Es utilizada por Hartmann [37] y Alcaraz [3]

3. Intercambio entre Actividades Consecutivas: Presentado por Hartman [35] este operador intercambia una actividad i con la actividad $i+1$ si no tienen ninguna relación de precedencia. Es utilizado por Alcaraz & Maroto [1] y Valls et al.[99], [98].
4. Movimiento a la Derecha: Fue presentado por Hartman [38]. Para implementarlo se recorre la lista de actividades de izquierda a derecha y con una probabilidad determinada se intercambia una actividad i con una actividad h a su derecha.
   Si se considera la lista de actividades $\lambda = \{j_1, \ldots, j_i, \ldots, j_h, \ldots, j_n\}$. Se quiere aplicar este operador a la actividad j_i. La actividad j_i puede ser movida luego de cualquier posición $h \in \{i+1, \ldots, n\}$, lo que implica que la nueva lista de actividades sería $\lambda' = \{j_1, \ldots, j_{i-1}, j_{i+1}, \ldots, j_h, j_i, j_{h+1} \ldots, j_n\}$. El cambio es mantenido si la secuencia resultante es una solución factible al problema.
5. Intercambio: Presentado por Debels & Vanoucke [24]. Aunque no se detalla su implementación, en términos generales consiste en intercambiar dos actividades cualquiera dentro de la lista de actividades. Lo más interesante de su enfoque es que en lugar de que el operador se ejecute según una probabilidad, el operador se aplica siempre que dos individuos vienen de padres que no se consideran diferentes.
6. Mutación Aleatoria: Uno de los más utilizados en el MRCPSP. Según una probabilidad de P_{mut} se modifica la lista de actividades o de modos de manera aleatoria. Utilizado por Mori[73] Özdamar [75], Hartman [37] y Van Petghem [104].

2.5.6. Reemplazo

Otra manera de introducir diversidad a la población es reemplazar los individuos ya existentes por individuos generados de manera aleatoria. En [71] el 20 % de peores individuos de la población se reemplazan por nuevos individuos generados aleatoriamente. En [2] a cada generación se puede aplicar el reemplazo de sus individuos dada una probabilidad p_{reemp}; si se aplica el operador de reemplazo, cada individuo puede ser reemplazado según una probabilidad p_{exch}.

2.5.7. Sustitución

El reemplazo de individuos es el operador que permite introducir a los hijos generados por los operadores genéticos en la población. En los AG existen dos

maneras de hacerlo: reemplazando toda la población o reemplazando solo unos cuantos individuos.

1. Sustitución Generacional: Históricamente los AGs han utilizado el reemplazo de la población completa (poblaciones no translapables), donde todos los individuos de la generación actual son reemplazados por los hijos obtenidos mediante los operadores descritos anteriormente.
2. Sustitución Generacional con Elitismo: Una de las desventajas de reemplazar toda la población con los hijos generados es la pérdida de las mejores soluciones. Para ello, se implementa el operador de elitismo, que pasa directamente los x mejores individuos de la población actual a la población siguiente. Por lo general, $x = 1$.
3. Sustitución Inmediata: También llamados AG de Estado Uniforme (*Steady State*, son aquellos AG donde los individuos que son generados por una pareja de padres entran directamente en la población y reemplazan a los miembros con menor aptitud, sin esperar a que se genere toda una población. Por lo general, solo se producen uno o dos hijos en cada iteración.
   Asociado a este tipo de algoritmos está el concepto de "*brecha generacional*", que denota la cantidad de superposición entre padres e hijos. Una brecha generacional alta implica poca o ninguna superposición (como en el caso de los AG Generacionales) y una brecha baja implica que padres e hijos están en la misma población y compiten entre si para ser seleccionados para generar nuevos individuos.
4. Combinación: Esta sustitución es utilizada en Mori & Tsheng [73]. La nueva generación se produce de la siguiente manera: Se pasan los P_1 individuos de la población anterior con mejor fitness. Esto es una estrategia elitista para conservar las mejores soluciones. Se producen P_2 individuos por medio del operador de cruce. Se producen P_3 individuos utilizando la mutación aleatoria de modos. Finalmente, se producen P_4 individuos utilizando el método de generación de la población inicial.

La Sustitución Generacional con Eltismo es la que se utiliza en todos los trabajos revisados de RCPSP y MRCPSP excepto el propuesto por Mori & Tsheng [73].

2.5.8. Funciones de Fitness utilizadas en el MRCPSP

Debido a la complejidad del problema, el hecho de generar una secuencia que sea factible ya consiste en si mismo en un problema complejo. Por ello, los algoritmos deben admitir soluciones no factibles y por tanto las funciones objetivo incluyen factores de penalización.

2.5.8.1. Duración del Proyecto

Cuando se manejan únicamente soluciones factibles en cuanto a relaciones de precedencia y uso de recursos no renovables, la minimización de la duración del proyecto es la función objetivo del algoritmo.

2.5.8.2. Función de penalización de Hartmann

En el algoritmo genético desarrollado por Hartmann para el MRCPSP es posible tener secuencias que no son factibles dado el uso de recursos no renovables. Sea $L_k^\nu(I)$ la capacidad sobrante de un recurso no renovable $k \in K^\nu$ con respecto a la asignación de modos μ del individuo I, que se define como:

$$L_k^\nu(I) = R_k^\nu(\mu_I) - \sum_{j=1}^{J} r_{j\mu(j)k}^\nu$$

Un valor negativo en la capacidad sobrante implica que la asignación de modos μ no es factible con respecto al recurso no renovable k. Puede entonces definirse la capacidad sobrante total como:

$$L^\nu(I) = \sum_{k \in K^\nu} |min(0, L_k^\nu(I)|$$

Se denota la duración de una secuencia relacionada con un individuo I como $C_{max}(I)$. Adicionalmente sea T la cota superior de la duración del proyecto dada por la suma de la duración máxima de las actividades. Ahora ya se puede definir la función objetivo de Hartmann como

$$F_{jmt} = \begin{cases} C_{max}(I) & \text{si } L^\nu(I) = 0, \\ T + L^\nu(I) & \text{en otro caso.} \end{cases}$$

Esta función es utilizada también en [3] y [104].

2.5.8.3. Función de penalización de Alcaraz

Al hacer un estudio de la función de penalización de Hartmann, Alcaraz et al.[3] hacen las siguientes observaciones:

- El valor de la función objetivo de un individuo no factible depende únicamente del exceso de requerimientos con respecto a los recursos no renovables, sin embargo la duración de la secuencia no se tiene en cuenta. Por tanto, dos individuos con el mismo exceso en el uso de recursos pero con diferente duración tendrán el mismo valor de la función objetivo.

- El límite superior T es mucho más alto que el valor de la duración de cualquier proyecto factible, de tal manera que la penalización es tan alga que los individuos no factibles tienen muy pocas probabilidades de ser seleccionados para la siguiente generación

Para superar estas debilidades tienen en cuenta el exceso en el uso de recursos no renovables del individuo en la nueva función objetivo para el algoritmo.

Sea C_{max}^{POP} la mayor duración de los individuos factibles de la población actual . Dicha duración se utiliza como cota superior de la duración de todos los individuos de la población actual que son factibles. $C_{max}(I) - CC_{min}$ es el incremento sobre la duración del proyecto determinada por la mínima ruta crítica del proyecto y $L^{\nu}(I)$ el exceso del uso de recursos no renovables, que representa el nivel de infactibilidad del individuo.

Con estos elementos se define la siguiente función de evaluación:

$$F_{jmt} = \begin{cases} C_{max}(I) & \text{si } L^{\nu}(I) = 0, \\ C_{max}^{pop} + L^{\nu}(I) + C_{max}(I) - CC_{min} & \text{en otro caso.} \end{cases}$$

Por lo tanto, la duración de los proyectos no factibles es siempre mayor que la de cualquier individuo factible y es menor que el valor propuesto por la penalización de Hartmann, aumentando las posibilidades de que un individuo no factible sea seleccionado para la siguiente generación.

Alcaraz [3] y Van Petenghem [104] hacen estudios comparativos de la función de penalización de Hartmann y Alcaraz con los métodos propuestos en cada trabajo y en ambos casos se encuentra que la función de Alcaraz obtiene mejores resultados que la de Hartmann.

2.6. Librerías de Prueba

Con el desarrollo de procedimientos heurísticos y exactos para los problemas de programación de proyectos surgió la necesidad de generar instancias para evaluarlos y compararlos. Durante muchos años los investigadores generaron sus propios problemas (Patterson(1984), Alvarez.Valdés (1989), Boctor (1990)). Esto generaba problemas de comparación entre métodos, debido a que los métodos desarrollados no evaluaban los mismos problemas.

A continuación describimos las dos librerías estándar de pruebas utilizadas en el área de Project Scheduling. En la sección 2.6.1 describimos la librería PSPLIB que es la más utilizada y que es la librería de referencia tanto en el RCPSP como en el MRCPSP. Luego en la sección 2.6.2 presentamos las características del conjunto de Boctor para el MRCPSP.

2.6.1. Librería de pruebas PSPLIB

Kolisch et al.[60] propusieron un generador para una clase general de problemas de programación de proyectos (ProGen[60]) el cual genera proyectos de forma aleatoria utilizando un diseño de experimentos factorial basado en dos conjuntos de parámetros. El primer conjunto consta de parámetros base que son iguales para cada conjunto de prueba:

- Número de actividades
- Número de modos en que cada actividad puede ejecutarse
- Número de recursos renovables existentes en el problema
- Disponibilidad máxima de cada recurso renovable
- Número de recursos no renovables existentes en el problema
- Disponibilidad máxima de cada recurso no renovable
- Número de sucesores de la actividad ficticia inicio
- Número de predecesores de la actividad ficticia fin
- Número de sucesores y predecesores de las actividades no ficticias
- Duración de las actividades

El segundo conjunto está conformado por los parámetros variables, es decir cambian de instancia a instancia:

- Complejidad de la Red (NC): Define le número medio de relaciones precedencia no redundantes por cada actividad.
- Factor de Recurso (RF): Refleja la proporción media del número de recursos diferentes que cada actividad no ficticia utiliza. Se aplica tanto a recursos renovables (RF_R) como no renovables(RF_{NR}). Si RF_R=1 quiere decir que la actividad utiliza todos los recursos renovables, por el contrario RF_R=0 significa que la actividad no utiliza ningún recurso renovable.
- Grado de restricción de los recursos (RS): Este parámetro expresa la relación entre la demanda de recursos de las actividades y la disponibilidad de los mismos; es decir mide la fortaleza de las restricciones o la escasez de los recursos. Este parámetro se calcula tanto para recursos renovables (RS_R) y no renovables (RS_{NR}).

2.6.1.1. Parámetros para el RCPSP

Los parámetros utilizados para la generación de problemas de RCPSP se presentan en la tabla 2.1

Las instancias donde RS=1 son triviales puesto que no tienen restricción en los recursos y por tanto son factibles las secuencias donde cada actividad es programada lo más pronto posible. Cada prueba tiene diez réplicas por lo que para J30, J60 y J90 hay 480 instancias y 600 instancias para el conjunto J120. Existen cuatro recursos renovables diferentes.

En diversos estudios con algoritmos heurísticos o exactos, se ha demostrado que el factor RS influye significativamente en la dificultad de las instancias y

Tabla 2.1. Factores variables utilizados en PSPLIB para el RCPSP

Factor	J10 - J60 - J90	J120
NC	1.50, 1.80, 2.10	1.50 , 1.80, 2.10
RF_R	0.25, 0.50, 0.75, 1.00	0.25, 0.50, 0.75, 1.00
RS_R	0.20, 0.50, 0.70, 1.00	0.1, 0.2, 0.3, 0.4, 0.5

cuanto menor es el valor de RS más difícil es de resolver dicha instancia([36], [52], [60]). Por tanto el conjunto J120 es el más complicado de resolver, no solo contiene los valores más restringidos para el parámetro RS y tiene el mayor número de problemas sin que ninguna de ellas sea trivial.

Los distintos métodos desarrollados se comparan por medio del porcentaje de desviación sobre la solución del límite inferior: Para las instancias del conjunto J30 es conocida la solución óptima y por tanto se calcula sobre este valor (%ADOS). Sin embargo para las instancias de los conjuntos J60 y J120 se calcula con respecto a la solución encontrada por el método del camino crítico puesto que el valor de la solución óptima no es conocido para todos los problemas (%ADLB). Para cada uno de los conjuntos se generan un máximo de 1.000, 5.000 y 50.000 soluciones

2.6.1.2. PSPLIB versión Multimodo

Para este caso, las actividades requieren de dos tipos diferentes de recursos renovables y dos no renovables. Cada actividad puede ser ejecutada en uno de tres modos diferentes. Los parámetros variables se presentan en la tabla 2.2 y todos tienen una NC=1.8.

Tabla 2.2. Factores variables utilizados en PSPLIB para el MRCPSP

Factor	J10	J12 - J14 - J16 - J18- J20 - J30
RF_R	0.25, 1.00	0.50, 1.00
RS_R	0.20, 0.50, 0.70, 1.00	0.25, 0.50, 0.75, 1.00
RF_{NR}	0.50, 1.00	0.50, 1.00
RS_{NR}	0.20, 0.50, 0.70, 1.00	0.25, 0.75, 1.00

Cada conjunto contiene 640 instancias, y existe entre un 14 % y un 16 % de instancias no factibles, las cuales no son consideradas al momento de resolver los problemas. La tabla 2.3 muestra el número de instancias factibles y el número promedio de modos factibles por actividad luego de aplicar el preproceso de Sprecher et al.[89] presentado al final de la sección 2.3.1.

Tabla 2.3. Instancias Multimodo PSPLIB

	J10	J12	J14	J16	J18	J20	J30
Instancias Factibles	536	547	551	550	552	554	552
Número promedio de modo por actividad	2.801	2.832	2.849	2.865	2.871	2.869	2.883

En la actualidad el conjunto de instancias con 20 actividades no factibles es el más difícil de resolver para el cual se conoce la solución óptima para todas las instancias que le conforman. Por ello, la calidad de las heurísticas se mide usualmente en términos de la desviación con respecto al óptimo.

Para el conjunto de 30 actividades no son conocidas todas las soluciones óptimas, por tanto la medida de eficiencia está basada en la desviación con respecto al camino crítico cuando se utiliza el modo con menor duración.

Estos conjuntos de prueba de máximo 30 actividades se consideran pequeños. Para probar la eficiencia del nuevo método se generan dos conjuntos adicionales de 60 y 120 actividades cada uno mediante el generador de instancias ProGen[60] con los parámetros del conjunto J30. Se generan un total de 560 instancias en cada uno de ellos. Estos problemas pueden ser descargados de la página web http://users.dsic.upv.es/grupos/gps.

2.6.2. Conjunto de Boctor

Boctor[12] presentó este conjunto de problemas en 1993. Está conformado por 240 instancias conformando dos subconjuntos. El primero contiene 120 instancias de 50 actividades no ficticias y el segundo contiene 120 instancias de 100 actividades no ficticias.

La generación de cada subconjunto se hizo de la siguiente manera. Primero se generaron de manera aleatoria 20 redes de actividades, cada una de las cuales tiene en promedio dos actividades sucesoras inmediatas. Luego, cada red se utilizó para generar seis problemas diferentes con uno, dos o cuatro tipo de recursos diferentes, los cuales son todos renovables.

El uso de recursos varía entre 1 y 5, con la restricción que el promedio ponderado del consumo de recursos es similar para todos los modos de ejecución.

Para cada actividad j, el número de modos de ejecución (M_j) es derivado de una distribución uniforme de 1 a 4. El tiempo de ejecución del primer modo (el más corto) varía entre 1 y 15 y la duración va aumentando en cada modo subsiguiente de manera proporcional a la fracción $15/M_j$.

El criterio de comparación para este conjunto es el porcentaje de desviación sobre la cota inferior (%ADLB). Dicha cota es la duración del proyecto calculada con el camino crítico en el modo más corto de las actividades, relajando la restricción de los recursos.

2.7. Conclusiones

En este capítulo hemos revisado la formulación de los problemas RCPSP, MRCPSP y la generación de programaciones robustas que son el foco de estudio en esta tesis doctoral. También hemos revisado los métodos de solución más relevantes para cada uno de ellos.

Los métodos de optimización exacta no pueden ser utilizados para resolver los problemas RCPSP y MRCPSP ya que el espacio de soluciones de una instancia mediana es muy amplio aún cuando se utilicen métodos de poda. Debido a la naturaleza del problema es preferible obtener varias soluciones con una calidad aceptable en poco tiempo, que una solución óptima en un tiempo no factible.

En el caso del problema RCPSP es necesario determinar el orden de las actividades para minimizar la duración de la programación. Se demostró que es un problema NP-duro[11]. Es uno de los problemas más estudiado en la literatura, debido a la diversidad de ámbitos en que puede aplicarse.

Los métodos más utilizados para su solución son los algoritmos genéticos, ya que son los que obtienen un mejor desempeño tanto en la calidad de las soluciones encontradas como en su eficiencia computacional. Los AG utilizan reglas de prioridad y un esquema de generación de secuencias para generar su población inicial. Una vez generada la solución factible se aplica el método de mejora FBI ya sea directamente o una modificación del mismo.

En el problema de modos múltiples (MRCPSP) hay dos decisiones diferentes que deben tomarse: el modo en que se ejecutará cada actividad y el orden de las actividades en la programación. Es un problema NP-duro, aún en instancias pequeñas [55].

Los algoritmos evolutivos son los que mejores resultados han dado en este caso. La asignación de modos se hace aleatoriamente y se utilizan diversos métodos de mejora enfocados principalmente en el cambio de una única actividad en cada iteración. Al permitir soluciones no factibles es necesario tener una función de evaluación que penalice dicha infactibilidad.

El problema de la generación de soluciones robustas se aborda desde dos perspectivas complementarias. Por un lado es necesario tener un método proactivo que genere una programación lo más estable posible frente a las eventualidades que puedan surgir en la ejecución. Una vez puesto en marcha el proyecto, se hace necesario tener un procedimiento reactivo que reprograme las actividades cuando sea necesario.

En los métodos proactivos se hace necesario tener medidas que sirvan de indicador de la estabilidad de la secuencia antes de que esta se ejecute. Una vez ejecutada la programación en un entorno dinámico es necesario determinar la estabilidad de las actividades y el nivel de cumplimiento en la finalización del proyecto global. En la actualidad los métodos proactivos son en su mayoría heurísticos basados en el impacto de la actividad en la estabilidad de la programación.

3

Propuesta y Evaluación de Métodos de Solución para el RCPSP

3.1. Introducción

Este capítulo está dedicado a los métodos que hemos desarrollado para el RCPSP. Tal como se mencionó en el capítulo anterior es un problema que ha sido ampliamente estudiado en la literatura debido a los múltiples ámbitos de aplicación. Debido a ello, el desarrollo de métodos competitivos con aquellos reportados en la literatura es ya una tarea compleja. Debido a la generalidad de este problema los métodos desarrollados pueden ser de provecho para otros problemas dentro del área de scheduling.

Diseñamos e implementamos un método heurístico basado en el uso de reglas de prioridad. Una vez hallada la programación inicial, se aplica el método de mejora FBI de manera selectiva. Esto implica que el número de pasadas forward/backward depende de la calidad de la solución. De otra proponemos una familia de algoritmos genéticos que incorporan el uso selectivo del método de mejora FBI. En este caso se tiene en cuenta tanto la calidad de la solución inicial como la profundidad de la búsqueda.

Los métodos desarrollados en este capítulo se implementaron en C y fueron compilados con Microsoft Visual C++ v.6.0 de Microsoft. Para la evaluación se utilizó un ordenador bajo Windows XP, con un procesador Pentium de 3GHz y 1GB en RAM.

Este capítulo está estructurado de la siguiente manera. En la sección 3.2 se detallan los nuevos métodos propuestos para el RCPSP. En la sección 3.3 se hace la evaluación de los métodos propuestos mediante la solución de los problemas de la la librería estándar de pruebas PSPLIB. Finalmente en la sección 3.4 se compara el desempeño con los mejores métodos publicados.

3.2. Nuevos Métodos Desarrollados para el problema RCPSP

En el problema de la programación de proyectos con recursos estudiamos el impacto del método de mejora FBI de Tormos & Lova [94] sobre diferentes reglas de prioridad. Con los resultados obtenidos se propone la nueva heurística de múltiples pasadas. Esta heurística es el punto de partida para la propuesta de una familia de algoritmos genéticos.

3.2.1. Impacto del Método de Mejora y Nueva Heurística Adaptativa

El desarrollo de la heurística adaptativa es el resultado un proceso de refinamiento. Inicialmente se hace un estudio de las reglas de prioridad que de acuerdo con la revisión de la literatura tienen un mejor desempeño. Se analiza así mismo el impacto del método FBI y el efecto de la aleatorización de los mismos. Las reglas estudiadas son las reglas del 1 al 15 de la tabla 3.1).

Por otra parte, diseñamos una nueva regla de prioridad basada en el **nivel de programación de la actividad**. Esta regla (Número 16 en la tabla) aprovecha la flexibilidad de programación que tienen aquellas actividades que cuentan con pocos predecesores y sucesores totales.

Se calcula, entonces, el nivel máximo de programación de cada actividad. Este es un proceso iterativo que asigna el nivel a todas las actividades empezando por la actividad ficticia fin y terminando en la actividad ficticia inicio.

$$N(i) = \begin{cases} 0 & \text{si } i = n+1 \\ \max\{N(i), N(j)+1\} & \forall i \in IP_j \end{cases} \tag{3.1}$$

Una vez obtenido el nivel de la actividad ficticia inicio N(0) se obtiene el nivel máximo de programación de cada actividad de la siguiente manera

$$N\max(j) = N(0) - N(j) \qquad \forall j \in J \tag{3.2}$$

Esta regla da prioridad a las actividades cuyo nivel de programación está más cercano del inicio y permite que las actividades con más flexibilidad sean programadas al final de la secuencia. Ello con el fin de programar primero aquellas actividades con menor flexibilidad, aunque tengan mayores tiempos de finalización y así contrastarla con la regla de prioridad MinLFT, que es la que tradicionalmente obtiene mejor desempeño.

La heurística multipasada incorpora las reglas de prioridad con mejor desempeño y el método aleatorizado RBRS para la generación de secuencias base. Con base en la calidad (duración) de las mismas se determinan dos tipos de solución para los cuales se utilizará el método de mejora[96] :

- Solución buena (SB): aquella que es mejor o igual que el promedio de las secuencias generadas hasta el momento, incluyendo las que se generan por RBRS y por el método de mejora

ID	Regla de Prioridad	Nombre	Criterio
1	Demanda de Recursos	MinRec	Min
2	Tiempo mas tardío de inicio	MinLST	Min
3	Tiempo más tardío de finalización	MinLFT	Min
4	Etiqueta de Actividad	MinID	Min
5	Holgura Total	MinHolg	Min
6	Tiempo más temprano de comienzo	MinEST	Min
7	Tiempo más más temprano de finalización	MinEFT	Min
8	Duración de la Actividad	MinDur	Min
9	Demanda de Recursos	MaxREC	Max
10	Número de Sucesores Inmediatos	MaxSUC	Max
11	Duración de la Actividad	MaxDur	Max
12	Tiempo más tardío de inicio y finalización	MinLFTLST	Min
13	Número Total de Sucesores	MaxSucTot	Max
14	Trabajo Restante	MaxGPRW	Max
15	Trabajo Restante Total	MaxGPRWTot	Max
16	Nivel Máximo de Programación	MinNivMax	Min

Tabla 3.1. Reglas de prioridad utilizadas

- Solución muy buena (SMB): aquella que es mejor que (promedio-(promedio-mejor)/2)

A las soluciones buenas se aplicará un máximo de dos pasadas BF y a las soluciones muy buenas un máximo de tres pasadas BF. Cuando se dice un máximo de pasadas BF se entiende que si no hay mejora entre la solución inicial y la obtenida luego de la primera pasada o entre dos pasadas BF consecutivas se detiene el proceso.

El procedimiento heurístico diseñado se presenta en el Algoritmo 3.1. Para construir la solución inicial se propone primero hacer una heurística determinística multipasada con dos reglas de prioridad MinLFT y MaxSucTot y el generador de secuencias paralelo en las dos direcciones de programación. A cada secuencia factible generada se le aplica el método de mejora FBI. La mejor prioridad se guarda y es el punto de inicio para la segunda fase.

La segunda fase consiste en la generación de secuencias utilizando la prioridad que se conserva de la fase anterior y utilizando RBRS. Con base en pruebas preliminares se determina que para los problemas del conjunto J30 y J60 se utilizan los esquemas serie y paralelo y para J120 se utiliza solo el paralelo. La dirección de programación se va alternando a medida que se generan las secuencias. Finalmente el método FBI se aplica a la secuencia obtenida.

Podría pensarse que al aplicar el esquema paralelo para los problemas de 120 actividades se reducen las posibilidades de encontrar el óptimo puesto que el esquema paralelo produce secuencias del tipo sin retraso (non-delay). Sin embargo, como se aplica el método FBI que utiliza el esquema serie, la

Algoritmo 3.1 Heurística Adaptativa

Inicialización Nsched=0
FASE 1
MejorPrioridad← HeuristicaMultiPasada(MinLFT, MaxSucTot, P, B/F)
FASE 2
while Nsched $\leq$ MaxSched **do**
  if (nact $<$ 45) **then**
    RBRS(α, P/S, B/F, MejorPrioridad)
  end if
  if (45 $\leq$ nact $\leq$ 0) **then**
    RBRS(α, P/S, B/F, MejorPrioridad)
  end if
  if (nact $>$ 90) **then**
    RBRS(α, P, B/F, MejorPrioridad)
  end if
  if (nact $\leq$ 45) (and RBRS es BuenaSolucion) **then**
    Se aplica un Máximo de **Dos BF/FB** pasadas
  end if

  if (nact $>$ 45) and (RBRS es MuyBuenaSolucion) **then**
    Se aplica un Máximo de **Tres BF/FB** pasadas
  end if
end while
Salida MejorSoluciónGenerada

secuencia generada es activa y por lo tanto incluye al menos una solución óptima [53].

Al igual que en los casos anteriores, el mecanismo acelerador permite que el método desarrollado detenga la búsqueda al encontrar una duración del proyecto igual a la dada por el método del camino crítico.

3.2.2. Algoritmo Genético Híbrido

Si bien el algoritmo genético aprovecha las conclusiones obtenidas del estudio de las diferentes reglas de prioridad y de la nueva heurística adaptativa, incluye dos factores novedosos: el tratamiento del tamaño de la población y la búsqueda local. Los individuos se representan mediante una lista de actividades, que ha demostrado ser la manera más efectiva para generar buenas soluciones[35]. Adicionalmente se incluyen dos genes binarios para codificar el esquema de secuenciación (Serie o Paralelo) y la dirección de programación (Forward o Backward).

A continuación se detallan los aspectos más relevantes del algoritmo que se propone.

3.2.2.1. Generación de la Población Inicial

La generación de la población inicial es importante ya que es la base del proceso evolutivo. Si se tienen individuos bien adaptados desde el comienzo del algoritmo, los operadores genéticos podrán actuar con mayor eficiencia.

El proceso de generación de la población inicial consta de dos fases. En la primera fase se aplican de manera determinística las reglas de prioridad MinLFT, MinLST y MaxSucTot. Se utilizan los dos métodos de generación de secuencias y las dos direcciones de programación, generando así doce soluciones. Estas reglas se escogen debido a los buenos resultados obtenidos en el estudio de la sección 3.2.1. Se utiliza el método iterativo de mejora FBI

La segunda fase consiste en la generación de secuencias utilizando las tres prioridades y el método de RBRS. El valor de α se determina para cada tamaño de problema, con los valores obtenidos en la sección 3.3.2. El método de generación de la secuencia y la dirección de programación se seleccionan aleatoriamente. Como método de mejora se hace una pasada B o F.

3.2.2.2. Tamaño de la Población

Una de las mayores desventajas de los AG propuestos hasta ahora, en particular de aquellos que logran los mejores resultados, es que el tamaño de la población depende completamente del número de soluciones que se generarán y asumen que se conocen de antemano [23], [25], [38] [98]. Este supuesto no es realista ni se cumple en casos prácticos.

Sin embargo, es cierto que el tamaño de la población está relacionado directamente con el número máximo de soluciones generadas límite horario y relacionado de manera inversa con el número de actividades. Así pues, el uso de una gran población se evita, de forma similar a la mutación, la creación de una población homogénea, y esto se hace más importante para los casos de pocas actividades y con muchas soluciones generadas.

Para evitar estas desventajas, se proponen dos alternativas:

- Tamaño Fijo: Esto quiere decir que el tamaño de la población se fija al inicio del AG y se mantiene a lo largo de la búsqueda
- Tamaño Variable: En este caso, se quiere aumentar el tamaño de la población a medida que aumenta la profundidad de la búsqueda. El grado de profundidad de la búsqueda se mide por medio del número de soluciones generadas hasta ese momento.

Se establece la población inicial en $2000/nact$, donde $nact$ es el número de actividades del proyecto. Se incrementa el tamaño de la población en 25 % de la población inicial cada mil nuevas soluciones generadas. En la tabla 3.2 se muestran los tamaños para distintos números de soluciones generadas.

Como es previsible que un individuo así generado sea menos apto que aquel obtenido por medio de la evolución en las distintas generaciones, y por ello tiene una probabilidad muy baja de reproducirse, se propone que estos

Tabla 3.2. Número de individuos de la Población Dinámica según las soluciones generadas

Nact	Pob Ini	1000	5000	10000	20000	30000	40000	49000
30	67	83	150	233	400	567	733	883
60	33	42	52	65	81	102	127	159
120	17	21	38	58	100	142	183	221

individuos se cruzan obligatoriamente con individuos de la población actual en la siguiente iteración.

3.2.2.3. Operadores Genéticos

Los operadores de selección evaluados en este algoritmo genético son el de 2-torneo y jerarquía. Debido a la simplicidad y efectividad se utilizará el cruce de dos puntos con una probabilidad del 80 %. En cuanto a la mutación se utiliza la mutación por inserción de Boctor debido a sus buenos resultados y adicionalmente siempre genera soluciones factibles.

En cuanto a la probabilidad de mutación se probarán los valores estándar de 1 % y 5 %. También se desarrolla una función que establece el porcentaje de probabilidad proporcional a la semejanza de los hijos.

Para estimar la semejanza de dos individuos, se utiliza una medida de distancia basada en la similitud de los padres. Para ello se calcula el número de actividades que ocupan la misma posición i en la lista de actividades del padre (x_p) y de la madre (x_m) dividido por el número de actividades. Si los dos individuos son idénticos la semejanza será 1 y en caso contrario 0.

Como se estableció antes, se quiere que la mutación sea como máximo un 5 %. Para ello se hace una interpolación lineal basado en el nivel de semejanza de los individuos: la probabilidad de mutación será 5 % para individuos con semejanza de 1 y 0 % para individuos con semejanza=0. Se hace también una segunda prueba con una probabilidad de mutación de 2.5 % para individuos de semejanza=1 siguiendo el proceso ya descrito. La primera función se llamará Semejanza5 y la segunda Semejanza 2.5.

3.2.2.4. Método de Búsqueda Local

Tal como se evidenció en los diversos métodos heurísticos desarrollados, el procedimiento de mejora produce mejores resultados cuando la secuencia base es de mejor calidad. Por tanto se implementa un proceso de búsqueda local basado en la calidad de la solución.

Los resultados de la heurística adaptativa evidencian un estancamiento en el algoritmo a partir de las 5.000 soluciones. Para evitar este fenómeno en el AG que se propone, consideramos también el grado de profundidad de la búsqueda a la hora de definir el número de pasadas BF que se deben

Algoritmo 3.2 Búsqueda Local Adaptativa

```
if solución es MuyBuenaSolucion then
  if totsol ≤ 5000 then
    BL: Máximo DOSBF
  else
    BL: Máximo TRESBF
  end if
else
  if solución es BuenaSolucion then
    if totsol ≤ 5000 then
      BL: B/F
    else
      BL: UNBF
    end if
  else
    No BL
  end if
end if
```

hacer. Proponemos entonces que entre más avanzada esté la búsqueda (mayor número de soluciones generadas), el número máximo de pasadas BF sea mayor. En el algoritmo 3.2 se presenta la búsqueda local adaptativa.

3.3. Evaluación de los métodos propuestos

Esta sección está dedicada a la evaluación de los métodos desarrollados para la solución del problema RCPSP descritos en la sección 3.2. Para evaluar su eficiencia resolvemos los problemas de la librería estándar de pruebas PSPLIB descrita en la sección 2.6.1. Finalmente comparamos su desempeño con los métodos con mejores resultados presentados en la literatura.

Para el conjunto J30 se conoce la solución óptima de todos los problemas. Para calcular la eficiencia de los métodos desarrollados para este conjunto se calcula el porcentaje de desviación media. Si C_{max} es la solución obtenida por el método y C^* la solución óptima, la desviación porcentual se calcula de la siguiente manera:

$$\%\ \text{DOS} = \frac{C_{max} - C^*}{C^*} * 100 \tag{3.3}$$

La desviación porcentual media sobre la solución óptima (%ADOS) es el promedio de %DOS de todos los problemas del conjunto.

Puesto que el valor de la solución óptima no es conocido para todos los problemas de los conjuntos J60 y J120 se calcula la cota inferior de la duración del proyecto. Esta cota es la solución encontrada por el método del camino crítico relajando las restricciones de los recursos. Esta medida la llamamos Desviación Porcentual media sobre la cota inferior (%ADLB). Para cada uno de los conjuntos se generan un máximo de 1.000, 5.000 y 50.000 soluciones.

3.3.1. Impacto del Método de Mejora FBI

En esta sección se quiere determinar el impacto del método de mejora FBI presentado en la sección 2.3.3. Para ello se aplica el método de mejora luego de ejecutar cada una de las reglas de la tabla 3.1. Se utilizan los esquemas de decodificación serie y paralelo y las direcciones de programación serie y forward, generando así 48 diferentes heurísticas.

Si la heurística inicial tiene dirección Forward el método hará primero una pasada Backward y luego una Forward (FB); por el contrario si la heurística tiene dirección Backward el método de mejora aplicará primero una pasada Forward y luego una Backward (BF) hasta que en dos pasadas consecutivas BF (FB) no haya mejora.

El método de mejora FBI es más rápido que la generación de una secuencia factible, pues no necesita crear el conjunto de actividades elegibles, sino que selecciona las actividades en el mismo orden en que aparecen en la secuencia factible. Sin embargo, cada pasada Backward o Forward se cuenta como una secuencia generada.

Reglas de prioridad determinísticas

En la figura 3.1 se comparan las heurísticas basadas en una regla de prioridad. Las heurísticas 1 a 16 son las generadas por las reglas de prioridad decodificadas con el esquema serie y dirección forward; las comprendidas entre 17 y 32 se generan utilizando el esquema serie backward, las siguientes 16 son generadas mediante el esquema paralelo forward y las últimas mediante el esquema paralelo backward.

En general, las heurísticas generadas al incluir el método de mejora FBI siguen el mismo patrón que las heurísticas iniciales. Por ello se puede inferir que se obtendrán mejores resultados al aplicar el método de mejora a heurísticas que tengan mejor desempeño. Por otra parte se observa como el método de mejora homogeniza los resultados, es decir las variaciones entre reglas de prioridad son menores a cuando se utiliza únicamente la regla de prioridad.

De otro lado, el método paralelo encuentra en general mejores soluciones que el método serie. Esta diferencia se hace aún más importante en los proyectos de gran tamaño (J120). Es también significativo, como el método paralelo (backward y forward) tiene un desempeño menos disímil que el presentado por las dos direcciones del método serie.

Las regla de prioridad MinLST, MinLFT, LSTLFT y MaxSucTot obtienen los mejores resultados con independencia del esquema generador o la dirección de programación.

Reglas de prioridad aleatorizadas

Las heurísticas presentadas en la sección anterior son determinísticas y por tanto devuelven siempre una única solución para cada problema, es decir

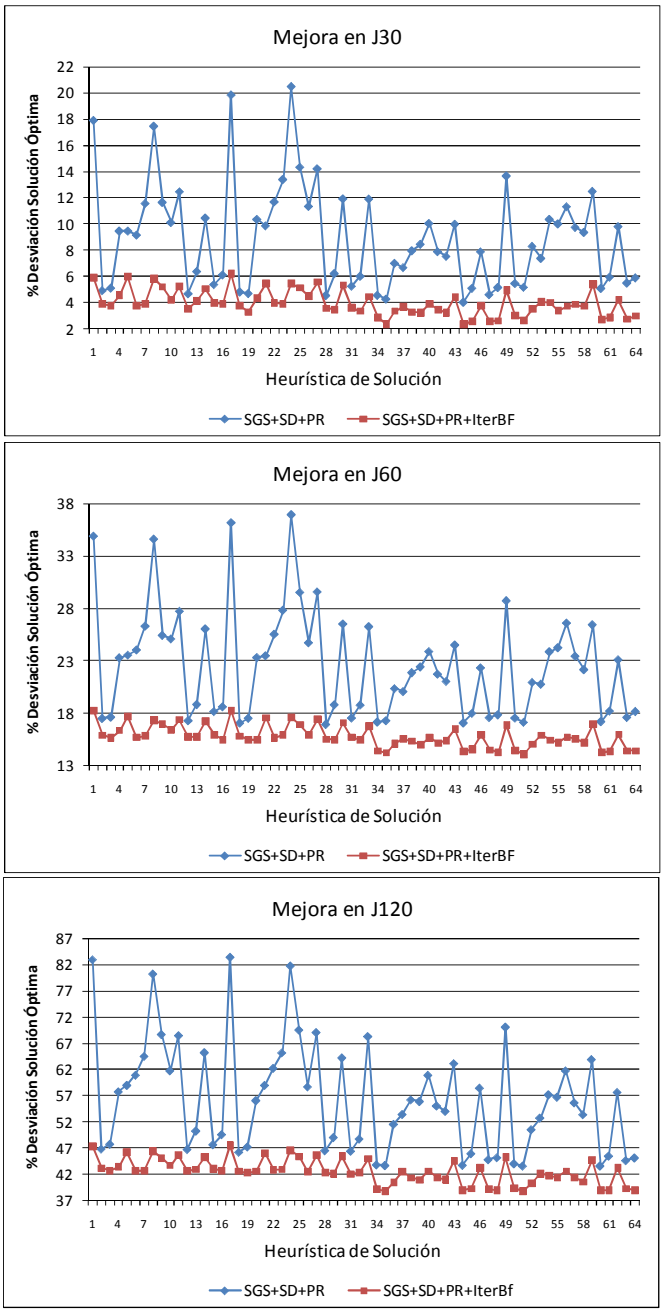


Figura 3.1. Mejora de Resultados de las Heurísticas

obtienen siempre los mismos resultados independientemente del número de veces que se ejecuten. Por tanto se quiere valorar el impacto que tiene incluir un método de aleatorización en los métodos descritos anteriormente.

Como método de aleatorización se utiliza el RBRS descrito en la sección 2.3.2.1, con ello la heurística obtiene diferentes resultados cada vez que se ejecuta. En este caso utilizamos un valor de $\alpha = 1$ y $\epsilon = 1$.

El impacto del método de mejora se presenta gráficamente en la figura 3.2 donde se observa como, al igual que en el caso determinístico, las reglas de prioridad que tienen mejores resultados antes de usar el método de mejora, son las que obtienen en mejores resultados al implementarlo.

El efecto del muestreo aleatorio tiene un mayor impacto sobre el conjunto J30, donde los resultados tienen la mayor mejora con respecto a los resultados de la heurística determinística. Por el contrario en el conjunto J120 la mejora es casi inexistente. Esto permite concluir que en un problema pequeño (pocas actividades) es mejor aumentar la aleatoreidad de la heurística, mientras que para un problema grande es mejor intensificar la búsqueda partiendo de una solución buena y por tanto el método debe ser menos variable.

Las reglas de prioridad MinLFT, LFTLST y MaxSucTot tienen los mejores resultados, en particular si se decodifican con el método paralelo.

3.3.2. Heurística Adaptativa

Con base en los resultados de las secciones anteriores, donde se evidencia que el método de mejora tiene un mayor impacto cuando la programación inicial es mejor, se propone la nueva heurística donde la cantidad de ciclos de mejora BF depende de la calidad de la solución inicial.

Para construir la solución inicial se ejecuta una heurística determinística con dos reglas de prioridad MinLFT y MaxSucTot y el generador de secuencias paralelo en las dos direcciones de programación. A cada secuencia factible generada se le aplica el método de mejora FBI. La mejor prioridad se guarda y es el punto de inicio para la segunda fase.

La segunda fase consiste en la generación de secuencias utilizando la prioridad que se conserva de la fase anterior y utilizando RBRS. Con base en los resultados de la sección 3.3.1 se determina que para los problemas del conjunto J30 y J60 se utilizan los esquemas serie y paralelo y para J120 se utiliza solo el paralelo. La dirección de programación se va alternando a medida que se generan las secuencias. Finalmente el método FBI se aplica a la secuencia obtenida, tal como se puede observar en el algoritmo 3.1.

El único parámetro que hace falta ajustar es el valor de α en el RBRS. Este parámetro permite controlar la cantidad de aleatoreidad introducida en el momento de seleccionar la actividad en el conjunto de decisión (actividades elegibles). Para cada conjunto de problemas se evaluaron diferentes valores tal como se resume en la tabla 3.3.

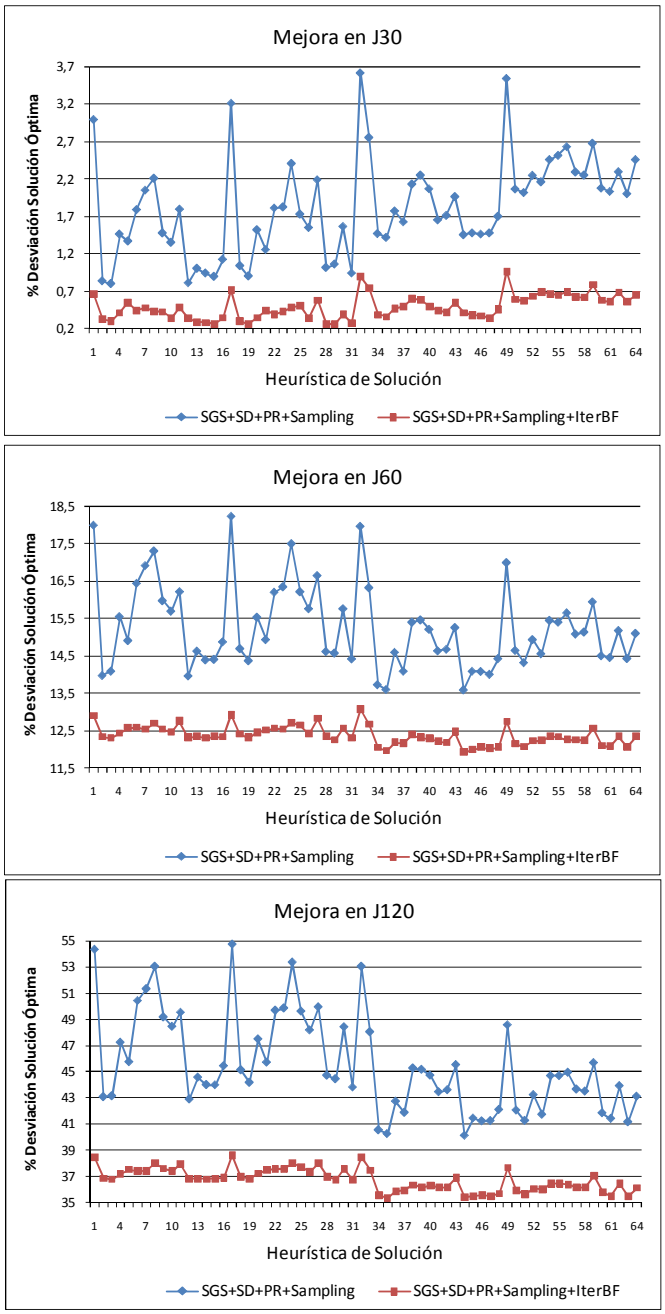


Figura 3.2. Mejora de las programaciones utilizando el método aleatorizado

Tabla 3.3. Impacto de α en RBRS + FBI

Conjunto J30				Conjunto J60				Conjunto J120			
α	SGS	1000	5000	α	SGS	1000	5000	α	SGS	1000	5000
1.5	S/P	0.25	0.11	3.0	S/P	11.87	11.56	9.0	P	34.84	34.31
2.0	S/P	0.20	0.08	3.5	S/P	11.82	11.55	10.0	P	34.79	34.23
2.5	S/P	0.24	0.11	4.0	S/P	11.84	11.56	11.0	P	34.81	34.26

Con base en estos resultados se selecciona $\alpha = 2$ para el conjunto de proyectos J30, $\alpha = 3{,}5$ para J60 y $\alpha = 10$ para J120. La configuración final de esta heurística se describe en el algoritmo 3.1.

3.3.3. Algoritmo Genético

Una vez evaluada la heurística propuesta, se determina que es la mejor entre los métodos heurísticos (no metaheurísticos) según lo reportado en la literatura. El siguiente paso es incluir el método de búsqueda local selectivo en un algoritmo genético básico. Evaluamos la estrategia de reemplazo de la población y el tamaño de la misma.

3.3.3.1. Algoritmo genético con sustitución poblacional y población estática

En primera instancia se hacen pruebas con la estrategia de reemplazo poblacional. Los factores a probar son: Operador de Selección, Operador de Mutación y Búsqueda Local.

Se establece un tamaño de población de 2000/nact, es decir 67, 33 y 17 individuos para los conjuntos J30, J60 y J120 respectivamente.

Evaluamos dos operadores de Selección: 2-torneo y Jerarquía. Estos métodos son efectivos en la prevención de la convergencia prematura de las soluciones a la vez que mantienen la presión selectiva durante la evolución.

El operador de mutación es el de inserción de Boctor y se evalúan dos niveles de uso común 5 % y 1 %. El procedimiento de mejora es el descrito en la sección 3.2.2.4. Los resultados obtenidos se presentan en la tabla 3.4

Tabla 3.4. %ADLB según el operador de selección.

	1000			5000			50000		
	J30	J60	J120	J30	J60	J120	J30	J60	J120
2T	0.15	11.42	34.29	0.05	11.03	33.31	0.04	10.84	32.54
Jerarquía	0.14	11.56	34.19	0.05	11.25	33.37	0.02	10.95	32.54

En cuanto al operador de mutación no hay duda en la literatura que la mutación por inserción de Boctor [14] es el más adecuado para el RCPSP. Debido a esto se han probado diferentes probabilidades de mutación: 1 % y 5 %. Adicionalmente se prueba la probabilidad asignada por la función de semejanza porcentual. Los resultados al aplicar estos cuatro valores de mutación se presentan en la tabla 3.5

Tabla 3.5. %ADLB según la probabilidad de mutación.

	1000			5000			50000		
	J30	J60	J120	J30	J60	J120	J30	J60	J120
1 %	0.16	11.47	34.07	0.10	11.20	33.26	0.08	11.06	32.60
5 %	0.15	11.42	34.29	0.05	11.03	33.31	0.04	10.84	32.54
Semejanza5	0.16	11.44	34.07	0.09	11.11	33.12	0.05	10.94	32.40
Semejanza2.5	0.17	11.46	34.17	0.09	11.20	33.28	0.06	11.05	33.28

En cuanto a los operadores de selección no hay dominancia por parte de ninguno. Se observa el impacto positivo de tasas cercanas al 5 % en la mutación, en particular en búsquedas largas y para problemas del conjunto J120.

3.3.3.2. Algoritmo genético con sustitución poblacional y población dinámica

Como resultado de la sección anterior se tiene que en las primeras 5000 iteraciones hay casi tanta mejora como en las siguientes 45000. Por tanto se quiere incluir un mecanismo que a medida que se progresa en la búsqueda inyecte diversidad a la población sin perder la calidad obtenida hasta el momento. Se quiere estudiar el impacto de una población dinámica en los resultados del AG.

Inicialmente, se prueban los operadores de selección (2Torneo y Jerarquía). Los resultados son los de la tabla 3.6.

Tabla 3.6. %ADLB según el operador de selección.

	1000			5000			50000		
	J30	J60	J120	J30	J60	J120	J30	J60	J120
2T	0.16	11.47	34.07	0.08	11.16	33.30	0.05	10.99	32.28
Jerarquía	0.14	11.56	34.19	0.06	11.25	33.42	0.02	11.02	32.91

Al trabajar con el operador 2Torneo los nuevos individuos tienen muy bajas probabilidades de sobrevivir. Entre más avanzada vaya la búsqueda se

eliminarán más pronto, pues son mucho menos adaptados que los individuos evolucionados.

A pesar de ello, el operador 2Torneo produce mejores resultados que el de Jerarquía para los conjuntos J60 y J120 y la diferencia se va incrementando conforme pasan las generaciones.

Estos resultados indican que no es la diversidad introducida lo que hace que el algoritmo mejore con respecto a lo obtenido en el caso de población estática. Lo que mejora los resultados de la búsqueda es que hay más individuos muestreando el espacio de solución y esto aumenta las probabilidades de mejorar los resultados.

Los resultados de las diferentes probabilidades de mutación están en la tabla 3.7

Tabla 3.7. %ADLB según el operador de mutación.

	1000			5000			50000		
	J30	J60	J120	J30	J60	J120	J30	J60	J120
1 %	0.16	11.47	34.07	0.08	11.16	33.30	0.05	10.99	32.28
5 %	0.15	11.42	34.00	0.06	11.01	33.16	0.02	10.87	32.54
/20	0.16	11.44	34.07	0.07	11.11	33.05	0.05	10.91	32.07
/40	0.17	11.46	34.17	0.09	11.18	33.22	0.06	10.96	32.02

En este caso se evidencia como la probabilidad de selección calculada por medio de la función de semejanza porcentual tiene mejores resultados en el conjunto J120. También se evidencia como para búsquedas largas es mejor considerar un porcentaje de mutación bajo y para búsquedas cortas un porcentaje más alto.

Si se compara con los resultados obtenidos para población estática, la mejora puede considerarse inexistente.

3.3.3.3. Algoritmo genético con sustitución inmediata y población dinámica

Finalmente se implementa un algoritmo genético con una estrategia de sustitución inmediata. Esta estrategia permite que los nuevos individuos sean incorporados directamente a la población actual, permitiendo que el algoritmo encuentre mejores soluciones de una manera más rápida. Para evitar la convergencia prematura los individuos generados por los operadores genéticos sólo entran a la población si su lista de actividades y su fitness es diferente al de todas los individuos que se encuentran en ese momento en la población.

Se evalúan los operadores 2Torneo y Jerarquía, los resultados están en la tabla 3.8. Al compararse con los resultados obtenidos previamente, se observa una mejora en los resultados con 5.000 y 50.000 iteraciones. En este caso, el operador de Jerarquía obtiene mejores resultados que en casos anteriores.

Tabla 3.8. %ADLB según el operador de selección.

	1000			5000			50000		
	J30	J60	J120	J30	J60	J120	J30	J60	J120
2T	0.17	11.43	33.94	0.05	10.99	32.78	0.02	10.80	31.77
Jerarquía	0.17	11.46	34.04	0.04	11.01	32.81	0.02	10.74	31.63

Los resultados obtenidos con las diferentes probabilidades de mutación están reportados en la tabla 3.9. De los tres conjuntos de pruebas, éste es en donde las diferentes probabilidades obtienen resultados más disímiles entre si en el conjunto J120.

Tabla 3.9. %ADLB según la probabilidad de mutación.

	1000			5000			50000		
	J30	J60	J120	J30	J60	J120	J30	J60	J120
1 %	0.20	11.43	33.97	0.07	11.02	32.84	0.03	10.78	32.00
5 %	0.17	11.43	33.94	0.05	10.99	32.78	0.02	10.80	31.77
/20	0.20	11.46	34.11	0.05	11.04	33.04	0.01	10.86	32.35

Hasta este momento los cálculos para definir las soluciones como “soluciones buenas” y “soluciones muy buenas” se hacen teniendo en cuenta todas las soluciones generadas en cada punto de evaluación. Esto quiere decir que al final del algoritmo las 50.000 soluciones generadas son la base del promedio.

Se quiere valorar el impacto de incluir en el promedio únicamente la solución que se obtiene luego de la aplicación del método de mejora de las secuencias. Los resultados están en la tabla 3.10.

Tabla 3.10. %ADLB según el operador de selección.

	1000			5000			50000		
	J30	J60	J120	J30	J60	J120	J30	J60	J120
Todos	0.17	11.43	33.94	0.05	10.99	32.78	0.02	10.80	31.77
Selectivo	0.16	11.43	33.71	0.04	10.96	32.57	0.01	10.81	31.65

Este criterio genera resultados mejores para el conjunto J120 sin desmejorar los de J30 y J60. Esto se debe a que la aplicación del método de mejora es más selectivo debido al aumento del valor promedio de las soluciones. Este aumento se debe a dos factores: al contar sólo con la solución final de cada

programación la calidad se incrementa, de otra parte el número de soluciones que se cuentan disminuye.

3.3.3.4. Algoritmo genético con población estática y sustitución inmediata

Luego de los resultados anteriores, en particular al observar que no hay mejora entre la población dinámica y la población estática, se quiere probar el algoritmo de sustitución inmediata y el promedio selectivo con una población estática. Los resultados de la tabla 3.11 al compararse con los de la tabla 3.10 indican que tener poblaciones estáticas o dinámicas no influyen la calidad del algoritmo.

Tabla 3.11. %ADLB según el tamaño de población.

	1000			5000			50000		
	J30	J60	J120	J30	J60	J120	J30	J60	J120
2000/nact	0.16	11.43	33.71	0.04	10.96	32.57	0.01	10.81	31.65
80	0.14	11.52	34.15	0.05	10.93	32.81	0.02	10.72	31.19

3.4. Comparación

En esta sección presentamos en detalle los resultados de la heurística adaptativa y de dos de los algoritmos genéticos propuestos. Para evaluar la eficiencia de estos algoritmos se ha complementado los resultados reportados en [57] con los que se han publicado a partir de ese momento. Los métodos están ordenados crecientemente con respecto a la solución hallada con 1.000 individuos.

3.4.1. Heurística Adaptativa

Para poder comparar los resultados obtenidos con trabajos anteriores se generan 1.000, 5.000 y 50.000 secuencias. Los resultados están en la tabla 3.12.

El efecto benéfico del método RBRS con las dos mejores prioridades y de la aplicación del método de mejora sobre buenas soluciones se observa desde ls primeras soluciones. Al generar solo 100 secuencias obtiene una desviación sobre el límite inferior de 0.62 %, 12.45 %, 35.95 % para J30, J60 y J120 respectivamente. Estos resultados son competitivos con los resultados de algunos de los mejores métodos con 1000 soluciones. (Tablas 3.15, 3.16 y 3.17).

Al compararlos con los resultados más recientes de la literatura, esta heurística está en los 15 primeros puestos para cualquiera de los tres conjuntos independientemente del número de soluciones generadas.

Tabla 3.12. Resultados Heurística Adaptativa

J30			
	1000	5000	500000
% ADOS	0.20	0.08	0.04
Desv. Est % ADOS	0.68	0.42	0.21
% Óptimos	90.00	95.42	97.29
Tiempo cpu (sg)	0.03	0.16	1.71
Secuencias Promedio	555	2756	27505
J60			
	1000	5000	500000
% ADLB	11.82	11.55	11.31
Desv. Est % ADLB	23.55	23.09	22.65
% Óptimos	61.25	61.46	61.46
Tiempo cpu (sg)	0.07	0.40	4.11
Secuencias Promedio	391	1935	19279
J120			
	1000	5000	500000
% ADLB	34.79	34.23	33.64
Desv. Est % ADLB	46.39	45.95	45.42
% Óptimos	26.17	26.83	27.83
Tiempo cpu (sg)	0.71	3.71	38.85
Secuencias Promedio	745	3682	36313

3.4.2. Algoritmo Genético

Una vez evaluadas las diversas combinaciones de parámetros, presentamos los dos algoritmos con mejores resultados. El algoritmo 1 es de población estática con 80 individuos a lo largo de toda la búsqueda y para los tres conjuntos de problemas. El algoritmo 2 es de población dinámica: la población inicial es de 2000/nact y cada 1.000 nuevas soluciones se incrementa en un 25 % de ese valor. Los parámetros de la configuración son:

Operador de Cruce	2Torneo
Probabilidad de Cruce	80 %
Operador de Mutación	Inserción
Probabilidad de Mutación	5 %
Búsqueda Local	Selectiva
Sustitución	Inmediata

Los resultados detallados del algoritmo 1 están en la tabla 3.13 y los del algoritmo 2 en la tabla 3.14.

La primera fila muestra el promedio de la desviación porcentual sobre la solución del límite inferior (%ADLB), la segunda muestra la desviación estándar del valor anterior. En la tercera fila se presenta el porcentaje de soluciones que obtienen el valor del límite inferior. La cuarta fila muestra el

tiempo promedio que tarda el algoritmo y por último se muestra el número de soluciones promedio generadas.

Tabla 3.13. Resultados Algoritmo Genético 1: Población Estática

J30			
	1000	5000	500000
% ADOS	0.14	0.05	0.02
Desv. Est % ADOS	0.52	0.29	0.23
% Óptimos	91.88	97.08	98.75
Tiempo cpu (sg)	0.02	0.11	1.10
Secuencias Promedio	554	2754	27505
J60			
	1000	5000	500000
% ADLB	11.52	10.93	10.72
Desv. Est % ADLB	22.96	21.85	21.38
% Óptimos	61.46	61.67	61.67
Tiempo cpu (sg)	0.05	0.22	1.95
Secuencias Promedio	392	1931	19217
J120			
	1000	5000	500000
% ADLB	34.15	32.81	31.19
Desv. Est % ADLB	48.80	44.71	42.74
% Óptimos	27.00	28.67	29.67
Tiempo cpu (sg)	0.39	1.65	10.89
Secuencias Promedio	743	3603	35294

Estos resultados evidencian como el algoritmo en sus dos versiones evoluciona hacia mejores soluciones. Los tiempos de ejecución pueden considerarse como eficientes, pues en promedio se tarda menos de 15 segundos para generar 50.000 soluciones de un problema de 120 actividades.

La diferencia en los tiempos de cómputo entre la versión de población estática y población dinámica se debe a que en este último el procedimiento de generación e inserción de los nuevos individuos hace que haya que comparar cada individuo generado con los ya existentes en la población.

El algoritmo de población estática se ubica en séptimo lugar para el conjunto J30, quinto para J60 y cuarto para J120. El algoritmo de población dinámica es tercero para los conjuntos J30 y J120, mientras que para J60 se ubica en décima posición.

Tabla 3.14. Resultados Algoritmo Genético 2: Población Dinámica

J30			
	1000	5000	500000
% ADOS	0.16	0.04	0.01
Desv. Est % ADOS	0.58	0.26	0.11
% Óptimos	92.29	97.29	99.17
Tiempo cpu (sg)	0.03	0.16	2.65
Secuencias Promedio	554	2754	27507
J60			
	1000	5000	500000
% ADLB	11.43	10.96	10.81
Desv. Est % ADLB	22.76	21.85	21.53
% Óptimos	61.46	61.67	61.67
Tiempo cpu (sg)	0.06	0.26	3.30
Secuencias Promedio	392	19326	19177
J120			
	1000	5000	500000
% ADLB	33.71	32.57	31.05
Desv. Est % ADLB	45.34	44.24	43.24
% Óptimos	27.50	28.33	29.17
Tiempo cpu (sg)	0.40	1.53	13.36
Secuencias Promedio	739	3612	35602

3.5. Conclusiones

En este capítulo hemos realizado la propuesta de un método heurístico que combina las reglas de prioridad con mejor desempeño en el problema RCPSP y el uso selectivo del método de mejora FBI. Para mejorar el desempeño de las reglas de prioridad se introduce un grado de aleatorización que se afina según el tamaño del problema para que explore un área mayor o menor de la región factible. Los resultados de esta heurística son comparables con los obtenidos por algunos de los mejores métodos metaheurísticos.

Así mismo hemos diseñado y evaluado una familia de algoritmos genéticos que incluyen aspectos novedosos como el tamaño variable de la población, la sustitución inmediata de la población y un operador de mutación que se ejecuta según la semejanza de los padres. Con base en las pruebas realizadas se pueden plantear las siguientes consideraciones.

- En general los dos operadores de cruce tienen un desempeño equivalente, aunque el operador de jerarquía produce resultados un poco mejores que el 2Torneo cuando las búsquedas son largas.
- En cuanto a la mutación, tener probabilidades alrededor del 5 % produce en términos medios mejores resultados que probabilidades inferiores.

Tabla 3.15. %ADOS - Conjunto J30

	Heurística	SGS	Referencia	% ADOS.		
				1000	5000	50000
1	GAPS - RK	S	Mendes et al. [71]	0.06	0.02	0.01
2	GA, TS - path relinking	P/S	Kochetov and Stolyar[51]	0.10	0.04	0.00
3	GA - Decomposition		Debels & Vanoucke [25]	0.12	0.04	0.02
4	GA - SS - Static Pop	P/S	Cervantes et al. [18]	0.14	0.05	0.02
5	GA - SS - Dynamic Pop	P/S	Cervantes et al. [18]	0.16	0.04	0.01
6	HybridGA	P/S	Alcaraz & Maroto [2]	0.15	0.06	0.01
7	BPGA	S	Debels & Vanoucke [24]	0.17	0.06	0.02
8	Scatter Search	S	Debels et al. [23]	0.17	0.11	0.01
9	Adaptive RBRS BF/FB	P/S	Lova et al. [69]	0.20	0.08	0.04
10	GA - forw, backw	P/S	Alcaraz et al. [4]	0.25	0.06	0.03
11	Comb-RBRS BF/FB	P/S	Tormos & Lova [96]	0.25	0.13	0.05
12	GA - hybrid	S	Valls et al. [98]	0.27	0.06	0.02
13	RBRS BF	P/S	Tormos & Lova [95]	0.30	0.17	0.09
14	RBRS BF	P/S	Tormos & Lova [94]	0.30	0.16	0.07
15	GA - AL	S	Alcaraz & Maroto [1]	0.33	0.12	*
16	GA -FBI	S	Valls et al. [99]	0.34	0.20	0.02
17	GA - self-adapting	P/S	Hartmann [38]	0.38	0.22	0.08
18	SA - AL	S	Bouleimen & Lecocq [15]	0.38	0.23	*
19	TS - activity list	S	Klein [50]	0.42	0.17	*
20	Sampling - random	S	Valls et al. [99]	0.46	0.28	0.11
21	TS - activity list	S	Nonobe & Ibaraki [74])	0.46	0.16	0.05
22	GA - activity list	S	Hartmann [35]	0.54	0.25	0.08
23	Sampling - adaptive	P/S	Schirmer [84]	0.65	0.44	*
24	GA - late join	S	Coelho & Tavares [21]	0.74	0.33	0.16
25	Sampling - adaptive	P/S	Kolisch & Drexl [54]	0.74	0.52	*
26	Sampling - global	S	Coelho & Tavares [21])	0.81	0.54	0.28
27	Sampling - MLFT	S	Kolisch [53]	0.83	0.53	0.27
28	TS - schedule scheme	Related	Baar et al. [9]	0.86	0.44	*
29	GA - random key	S	Hartmann [35]	1.03	0.56	0.23
30	GA - priority rule	S	Hartmann [35]	1.38	1.12	0.88
31	Sampling-MLFT	P	Kolisch [53]	1.40	1.29	*
32	Sampling-WCS	P	Kolisch [53]	1.40	1.28	*
33	Sampling-random	S	Kolisch [52]	1.44	1.00	0.51
34	Sampling-random	P	Kolisch [52]	1.77	1.48	1.22
35	GA - problem space	Mod. P	Leon & Ramamoorthy [64]	2.08	1.59	*

- El operador basado en la semejanza porcentual tiene un mejor desempeño para el conjunto J120 si se compara con las probabilidades del 1 y 5 %.
- Incluir una población dinámica, al menos como se ha diseñado en este trabajo, no permite al algoritmo mejorar de una manera importante. Los resultados de pruebas comparables así lo demuestran.

Tabla 3.16. %ADLB - Conjunto J60

	Heurística	SGS	Referencia	% ADLB.		
				1000	5000	50000
1	GA - Decomposition		Debels & Vanoucke [25]	11.31	10.95	10.68
2	GA - SS - Dynamic Pop	P/S	Cervantes et al. [18]	11.43	10.96	10.81
3	BPGA	S	Debels & Vanoucke [24]	11.45	11.00	10.69
4	GA - SS - Static Pop	P/S	Cervantes et al. [18]	11.52	10.93	10.72
5	GA - hybrid	S	Valls et al. [98]	11.56	11.10	10.73
6	HybridGA B/F	P/S	Alcaraz & Maroto[2]	11.67	11.05	10.80
7	GA, TS - path relinking	P/S	Kochetov & Stolyar [51]	11.71	11.17	10.74
8	GAPS	S+RK	Mendes et al. [71]	11.72	11.04	10.67
9	Scatter Search	S	Debels et al. [23])	11.73	11.10	10.71
10	Adaptive RBRS BF/FB	P/S	Lova et al. [69]	11.82	11.85	11.31
11	Comb-RBRS BF/FB	P/S	Tormos & Lova [96]	11.88	11.62	11.36
12	GA - forw, backw	P/S	Alcaraz et al. [4]	11.89	11.19	10.84
13	RBRS BF	P/S	Tormos & Lova [95]	12.14	11.82	11.47
14	RBRS BF	P/S	Tormos & Lova [94])	12.18	11.87	11.54
15	Self-adapting GA	P/S	Hartmann [40]	12.21	11.70	11.21
16	GA	S	Valls et al. [99]	12.21	11.27	10.74
17	GA - activity list	S	Alcaraz & Maroto [1]	12.57	11.86	*
18	GA - activity list	S	Hartmann [35]	12.68	11.89	11.23
19	Sampling - random	S	Valls et al. [99]	12.73	12.35	11.94
20	SA - activity list	S	Bouleimen & Lecocq [15]	12.75	11.90	*
21	TS - activity list	S	Klein [50]	12.77	12.03	*
22	Sampling - adaptive	P/S	Schirmer [84]	12.94	12.59	*
23	TS - activity list	S	Nonobe & Ibaraki [74]	12.97	12.18	11.58
24	GA - late join	S	Coelho & Tavares [21]	13.28	12.63	11.94
25	GA - priority rule	S	Hartmann [35]	13.30	12.74	12.26
26	Sampling - adaptive	P/S	Kolisch & Drexl [54]	13.51	13.06	*
27	Sampling-MLFT	P	Kolisch [53]	13.59	13.23	12.85
28	Sampling-WCS	P	Kolisch [53]	13.66	13.21	*
29	TS - sched. Scheme	Related	Baar et al. [9]	13.80	13.48	*
30	Sampling - global	S	Coelho & Tavares [21]	13.80	13.31	12.83
31	Sampling-MLFT	S	Kolisch [53]	13.96	13.53	12.97
32	GA - problem space	Mod. P	Leon & Ramamoorthy [64]	14.33	13.49	14.33
33	GA - random key	S	Hartmann [35]	14.68	13.32	12.25
34	Sampling-random	P	Kolisch [52]	14.89	14.30	13.66
35	Sampling-random	S	Kolisch [52]	15.94	15.17	14.22

- La estrategia de reemplazo inmediato, aunque no es muy utilizada en AG para problemas de optimización muestra su efectividad, pues los mejores resultados se obtienen al implementarla en nuestro algoritmo.

El algoritmo genético de reemplazo inmediato al ser comparado con los métodos publicados hasta la fecha, se ubica entre los cinco mejores para los tres conjuntos de prueba.

Tabla 3.17. %ADLB - Conjunto J120

	Heurística	SGS	Referencia	% ADLB.		
				1000	5000	50000
1	GA - Decomposition		Debels & Vanoucke [25]	33.55	32.18	30.69
2	GA - SS - Dynamic Pop	P/S	Cervantes et al. [18]	33.71	32.57	31.05
3	GA - hybrid	S	Valls et al. [98]	34.07	32.54	31.24
4	GA - SS - Static Pop	P/S	Cervantes et al. [18]	34.15	32.81	31.19
5	BPGA	S	Debels & Vanoucke [24]	34.29	32.34	30.75
6	GA, TS - path relinking	P/S	Kochetov & Stolyar [51]	34.74	33.36	32.06
7	Adaptive RBRS BF/FB P/S	P/S	Lova et al. [69]	34.79	34.23	33.64
8	RBRS BF/FB	P/S	Tormos & Lova [96]	35.01	34.41	33.71
9	Population-based-FBI	S	Valls et al. [99]	35.18	34.02	32.81
10	Scatter Search	S	Debels et al. [23]	35.22	33.10	31.57
11	GA	S	Valls et al. [99]	35.39	33.24	31.58
12	RBRS BF	P/S	Tormos & Lova [95]	36.24	35.56	34.77
13	RBRS BF	P/S	Tormos & Lova [94]	36.49	35.81	35.01
14	GA - forw, backw	P/S	Alcaraz et al. [4]	36.53	33.91	31.49
15	Self-adapting GA	P/S	Hartmann [40]	37.19	35.39	33.21
16	Sampling - random	S	Valls et al. [99]	38.21	37.47	36.46
17	GA - activity list	S	Alcaraz & Maroto [1]	39.36	36.57	*
18	GA - activity list	S	Hartmann [35]	39.37	36.74	34.03
19	Sampling-MLFT	P	Kolisch [53]	39.6	38.75	37.74
20	Sampling-WCS	P	Kolisch [53]	39.65	38.77	*
21	Sampling - adaptive	P/S	Schirmer [84]	39.85	38.70	*
22	GA - priority rule	S	Hartmann [35]	39.93	38.49	36.51
23	GA - late join	S	Coelho & Tavares [21]	39.97	38.41	36.44
24	TS - activity list	S	Nonobe & Ibaraki [74]	40.86	37.88	35.85
25	Sampling - global	S	Coelho & Tavares [21]	41.36	40.46	39.41
26	Sampling - adaptive	P/S	Kolisch & Drexl [54]	41.37	40.45	*
27	SA - activity list	S	Bouleimen & Lecocq [15]	42.81	37.68	*
28	Sampling-MLFT	S	Kolisch [53]	42.84	41.84	40.63
29	GA - problem space	Mod. P	Leon & Ramamoorthy [64]	42.91	40.69	*
30	Sampling-random	P	Kolisch [52]	44.46	43.05	41.44
31	GA - random key	S	Hartmann [35]	45.82	42.25	38.83
32	Sampling-random	S	Kolisch [52]	49.25	47.61	45.60

4

Propuesta y Evaluación de Métodos de Solución para el MRCPSP

4.1. Introducción

En este capítulo presentamos los nuevos métodos que hemos desarrollado para el problema de programación de proyectos cuando las actividades pueden ejecutarse en uno de múltiples modos. Este problema es una extensión del RCPSP, su solución es compleja pues implica la toma de dos decisiones, cada una de las cuales constituye un problema NP-duro. Por un lado se hace necesario seleccionar el modo en que cada activad ha de ser ejecutada y de otra parte se debe determinar el orden de ejecución de las actividades.

Las principales aportaciones en este problema son el método de asignación de modos y la extensión del método de mejora FBI al caso multimodo. Estos procedimientos se integran en un algoritmo genético, para el cual se implementa una nueva función de evaluación de los individuos y un operador de mutación que ayuda a lograr la factibilidad de las soluciones que exceden la disponibilidad de los recursos no renovables.

Los métodos desarrollados en este capítulo se implementaron en C y fueron compilados con Microsoft Visual C++ v.6.0 de Microsoft. Para la evaluación se utilizó un ordenador bajo Windows XP, con un procesador Pentium de 3GHz y 1GB en RAM.

Las aportaciones en cada uno de los aspectos de solución del MRCPSP se describen en la sección 4.2. En la sección 4.3 se hace evalúan dichas propuestas mediante la solución de los problemas de la la librería estándar de pruebas PSPLIB y el conjunto de Boctor. Finalmente en la sección 4.4 se compara el desempeño con los mejores métodos publicados.

4.2. Aportaciones en el problema MRCPSP

En el caso de la programación de proyectos con recursos restringidos y múltiples modos de ejecución de las actividades, las aportaciones se agrupan en dos categorías.

La primera categoría hace referencia a las aportaciones en la solución del problema. Se propone la extensión del método de mejora de programaciones factibles FBI al caso multimodo y el diseño de un método de asignación de modos de ejecución a las actividades. En cuanto al método de mejora, éste se aplica a programaciones factibles y aprovecha la holgura libre de las actividades y la posibilidad de cambiar el modo de ejecución por uno más corto. Así mismo determinamos el esfuerzo computacional requerido por este método, ya que la comunidad científica que trabaja en este tema compara la eficiencia de los algoritmos mediante el número de soluciones generadas.

La segunda categoría hace referencia al desarrollo de un algoritmo genético y las novedades que se incluyen en él, tales como la generación de la población inicial, la función de penalización para las soluciones no factibles y el operador de mutación masiva.

4.2.1. Método de Mejora de Programaciones Factibles - MM-FBI

El método propuesto es una extensión del método FBI de Tormos & Lova [94] que fue presentado en la sección 2.3.3. Este método es ampliamente utilizado en los trabajos más recientes para la solución del RCPSP mediante métodos aproximados.

Al considerar la existencia de múltiples modos de ejecución de cada actividad, el diseño de un método de búsqueda local requiere que, además de la modificación de los tiempos de programación de las actividades sea también posible el cambio del modo de ejecución de tal manera que la duración total del proyecto se reduzca.

Una programación para el problema MRCPSP consiste en una asignación de modos M y una programación S, que se denota como (S,M). La asignación de modos es una tupla $M = \{m_1, \ldots, m_N\}$ que asigna a cada actividad j un único modo de ejecución m_j. La programación S es una tupla $S = \{s_1, \ldots, s_N\}$ que asigna a cada actividad j un único tiempo de inicio s_j.

La duración del proyecto factible puede ser reducida mediante la aplicación del Método de Mejora Multimodo Backward-Forward (MM-BF) o con el método de Mejora Multimodo Forward-Backward (MM-FB) en función de si la programación factible inicial ha sido obtenida en dirección Forward o Backward respectivamente.

Cada iteración de estos métodos consiste en una pasada Backward Multimodo y una pasada Forward Multimodo. La pasada Backward MultiModo (MM-B) ordena decrecientemente las actividades según su tiempo de finalización. La actividad i se reprograma en el modo factible con respecto a los recursos renovables y no renovables que le permita programarse en su máximo tiempo posible de inicio en la ventana temporal delimitada por

$$[S'_i, \min(S_j - d_{im})] \qquad m \in M_i, \quad \forall (i,j) \in A \tag{4.1}$$

donde S'_j es el tiempo de inicio de la actividad j en la programación factible actualizada. En caso de empates, se escoge el modo con menor duración. Por

lo tanto, este proceso implica un cambio de modo en la actividad i con el fin de aumentar la ventana de tiempo de sus actividades predecesoras.

Una vez todas las actividades han sido evaluadas, si $S^{min} = min(S_j) > 0$ implica que ha habido una reducción en la duración del proyecto de S^{min} unidades. En tal caso, el inicio programado de cada actividad es actualizado mediante la expresión $S_j = S_j - S^{min}$.

En el proceso para la pasada Forward Multimodo (MM-F)las actividades se ordenan en orden creciente de su tiempo de inicio. Cada actividad i es reprogramada en el modo que sea factible con respecto a los recursos renovables y no renovables que le permita programarse en el tiempo factible más temprano de finalización en la ventana temporal delimitada por

$$[\max(F'_j, S_i] \qquad \forall (i,j) \in A \tag{4.2}$$

donde F'_j s el tiempo de finalización de la actividad j en la programación factible actualizada. En caso de empates, se escoge el modo con la duración más corta con el fin de ampliar a ventana de mejora de las actividades sucesoras de la actividad i.

Para mantener la factibilidad de la programación con respecto a los recursos no renovables, cuando una actividad es reprogramada en cualquiera de las dos pasadas, MM-F o MM-B, únicamente se consideran los modos que mantienen dicha factibilidad.

El método MM-BF comienza con una secuencia factible obtenida con un SGS en la dirección Forward y cada iteración implica la aplicación de la pasada Backward Multimodo seguida de la pasada Forward Multimodo. Por otra parte, el método MM-FB parte de una secuencia factible obtenida con un SGS en la dirección Backward y cada iteración implica una pasada Forward Multimodo seguida de una pasada Backward Multimodo.

Por tanto cada pasada MM-FBI (MM-FB o MM-BF) obtiene una programación factible con una duración del proyecto que puede ser menor o igual a la de la programación base. Como es posible realizar iteraciones adicionales que pueden reducir aún más la duración, las pasadas MM-FBI se ejecutan iterativamente hasta que no haya ninguna mejora en dos ciclos consecutivos.

4.2.1.1. Esfuerzo computacional del Método MM-FBI

Uno de los temas más controvertidos a la hora de comparar métodos de solución del MRCPSP es el esfuerzo computacional requerido, ya que es la manera que la comunidad científica utiliza para la comparación de los diversos métodos propuestos. Este esfuerzo se ha medido de dos maneras: tiempo de cómputo y soluciones generadas.

El tiempo de procesamiento no permite hacer comparaciones directas de los métodos desarrollados, ya que con el cambio de alguna característica del ordenador se afecta el tiempo empleado en procesar el método desarrollado.

Al comparar el desempeño de un conjunto de algoritmos es indispensable hacer la comparación basándose en el mismo esfuerzo computacional sin que sea afectado por el tipo de ordenador utilizado. Con esta idea en mente, desde el trabajo de Hartmann & Kolisch [40] el desempeño de diferentes algoritmos se basa en el número de programaciones generadas para alcanzar un resultado.

Adicionalmente Kolisch & Hartmann [57] han especificado este enfoque aún más y han propuesto que el esfuerzo computacional para generar una programación debe corresponder a lo máximo a una asignación de tiempo de inicio por actividad, tal como se hace en un SGS.

Teniendo esto en cuenta, es obvio que el esfuerzo de un MM-FBI es mayor que el necesario para generar dos SGSs. Por tanto, se determina que el esfuerzo computacional para generar una MM-B o un MM-F es igual a la suma de las veces que cada actividad del proyecto ha obtenido un posible instante de inicio, dividido por el número de actividades del proyecto.

4.2.1.2. Ejemplo

Con el fin de ilustrar la técnica de mejora propuesta y luego el algoritmo desarrollado, se utiliza un proyecto de ejemplo con 10 actividades ficticias que requieren para su ejecución dos tipos de recursos renovables y dos tipos de recursos no renovables. Concretamente, el proyecto ejemplo utilizado es la instancia J1037_2, perteneciente a la serie J10 generados por Kolisch & Sprecher [59].

Este proyecto es uno de los más difíciles de resolver ya que todas las actividades requieren el uso de todos los recursos disponibles. En la tabla 4.1, se muestra para cada actividad sus sucesoras inmediatas, la duración y los recursos necesarios para cada modo.

Tabla 4.1. J1037_2 - Disponibilidad (R1,R2, NR3, NR4)=(12, 12, 37, 60)

Actividad	Sucesores Inmediatos	Modo 1					Modo 2					Modo 3				
			R		NR			R		NR			R		NR	
		d_{j1}	r_{j11}	r_{j12}	r_{j13}	r_{j14}	d_{j2}	r_{j21}	r_{j22}	r_{j23}	r_{j24}	d_{j3}	r_{j31}	r_{j32}	r_{j33}	r_{j34}
1	4, 5, 10	2	5	9	4	7	5	5	8	2	7	6	5	6	1	6
2	6	1	6	5	6	8	8	5	4	5	5	8	4	5	4	6
3	6, 10	1	4	9	9	5	8	3	4	6	1	8	1	4	2	3
4	8, 9	1	7	3	7	8	2	6	3	5	8	2	5	3	6	5
5	7	1	7	9	4	10	4	7	7	2	10	9	6	2	1	10
6	7, 8	1	9	5	3	6	1	9	4	3	7	5	9	3	3	6
7	9	2	9	5	9	8	2	9	6	7	8	3	9	4	4	8
8	---	1	9	9	10	6	5	9	8	6	6	9	9	8	4	5
9	---	2	7	2	4	9	9	7	2	4	6	10	3	1	3	3
10	---	3	5	10	8	10	6	4	10	5	7	7	4	9	3	7

Utilizando la selección de modos factible para la solución inicial que se presenta en la segunda columna de la tabla 4.2, se calcula el tiempo de comienzo de cada actividad mediante el método serie forward (Figura 4.1a). La duración del proyecto es de 38 unidades de tiempo y la programación factible así obtenida se convierte en el punto de inicio del método de mejora MM-FBI. Luego de hacer la pasada backward (MM-B), la finalización del

Tabla 4.2. Asignación de Modo inicial y tiempo de inicio en cada etapa del método de mejora MM-FBI

			MM-BFI			
	Solución Inicial Factible		MM-B		MM-F	
Actividad	Asignación de Modo	S_j	Asignación de Modo	S_j	Asignación de Modo	S_j
1	1	0	1	0	1	0
2	2	2	2	3	2	2
3	3	6	3	3	3	2
4	3	6	3	19	3	15
5	2	2	3	2	3	2
6	3	14	1	11	1	11
7	3	19	3	12	3	12
8	3	29	3	22	2	22
9	3	22	3	21	3	17
10	3	22	3	15	3	15

proyecto se reduce en 7 periodos de tiempo Figura 4.1b). Esta pasada implica el cambio del modo asignado a las actividades 5 y 6, tal como se refleja en la cuarta columna de la Tabla 4.2.

Cuando se aplica la pasada forward (MM-F), la duración del proyecto tiene una reducción adicional de 4 periodos de tiempo (Fig 4.1c). En esta pasada la actividad 8 cambia de modo. La asignación final se muestra en la sexta columna de la Tabla 4.1.

La solución final obtenida luego de aplicar una pasada MM-FBI tiene una duración de 27 periodos de tiempo, lo que significa que el makespan es 11 periodos de tiempo menor que la solución factible inicial. Esta duración corresponde a la duración óptima para esta instancia. Tal como se mencionó en la sección anterior, en otros problemas se pueden necesitar pasadas adicionales del método MM-FBI para obtener mayores reducciones del makespan.

4.2.2. Selección del Modo de Ejecución de las Actividades

El problema de la Selección del Modo de ejecución de las actividades es un problema NP-duro tal como demostró Kolisch [55]. Hasta ahora la mayor parte de los métodos aproximados para resolver el MRCPSP parten de una asignación aleatoria de modos de ejecución, con lo cual no se tiene ninguna

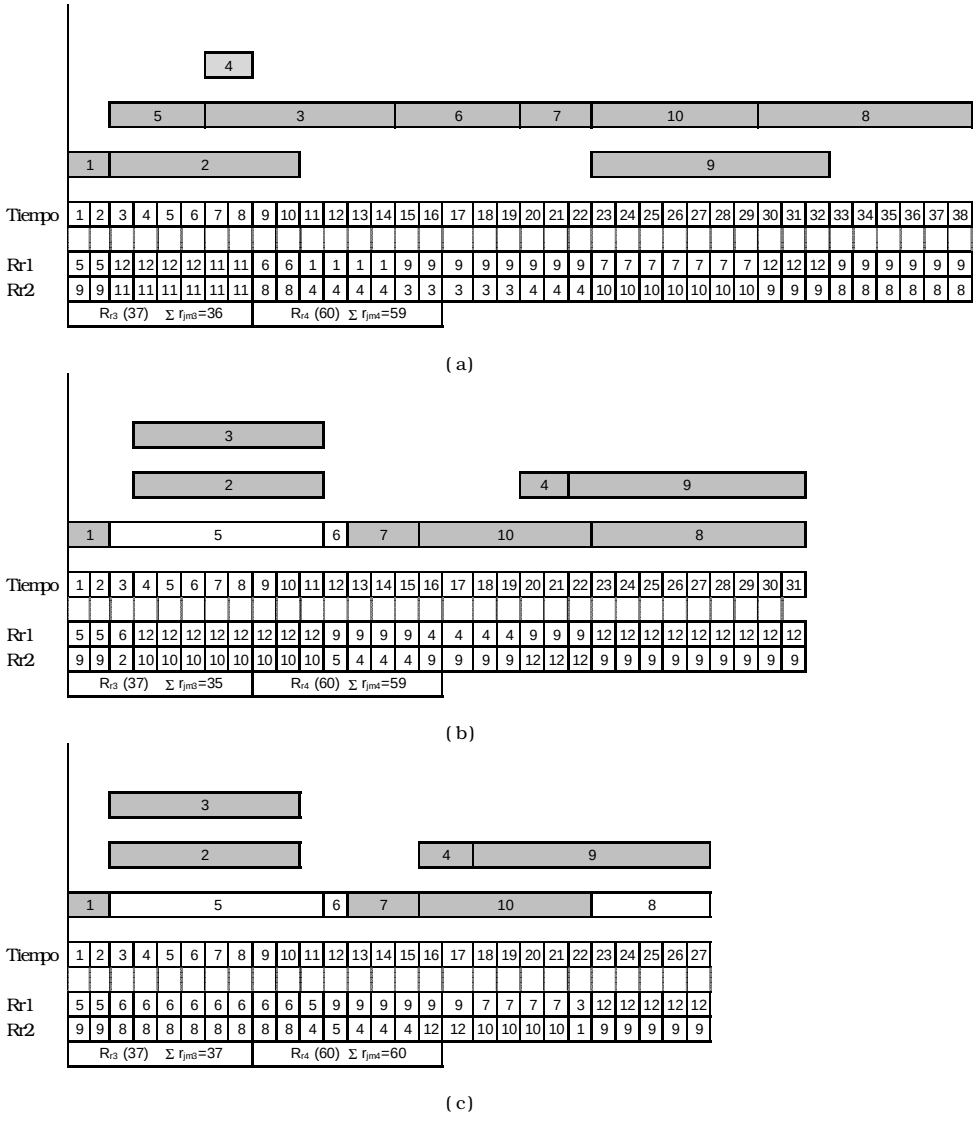


Figura 4.1. (a) Solución Inicial Factible (b) Solución luego de aplicar MM-B (c) solución luego de aplicar MM-F

garantía de la factibilidad de las soluciones iniciales. Por ello se propone un procedimiento para maximizar la probabilidad de obtener soluciones factibles en la solución inicial, de tal manera que el punto de partida para cualquier método de solución.

La probabilidad de obtener una asignación de modos que sea factible por medio de este procedimiento es bastante alta y no requiere ningún esfuerzo computacional adicional.

Este procedimiento ha sido llamado **Recursos Mínimos Normalizados**(MNR). Implica, para cada actividad, la selección del modo que minimice el valor normalizado:

$$NR_{jm} = \sum_{k \in R^v} \frac{r_{jmk}}{a_k} \tag{4.3}$$

Para ilustrar la aplicación de este criterio, la tabla 4.3 muestra los diferentes modos de ejecución de una actividad que utiliza dos recursos no renovables R_1 y R_2. Su disponibilidad total es $a_1 = 40$ y $a_2 = 20$. De acuerdo con el criterio MNR se escogería el modo 2.

Tabla 4.3. Ejemplo del criterio MNR

Modo	r_{jm1}	r_{jm2}	MNR
1	0	7	0/40 + 7/20=0.35
2	3	0	3/40 + 0/20=0.075
3	0	2	0/40 + 2/20=0.10

4.2.3. Algoritmo Genético Híbrido

Los dos métodos anteriores se incorporan a un algoritmo genético que es nuestra propuesta de solución integral al MRCPSP. El algoritmo 4.2 describe su estructura. A continuación se describen los nuevos elementos diseñados específicamente para el AG que se empleará en la resolución de este problema.

4.2.3.1. Evaluación de los individuos

La función de evaluación es de vital importancia porque con base en ella se determinan las probabilidades con las que un individuo podrá pasar a la siguiente generación. Frecuentemente la función de evaluación coincide con la función objetivo del problema. Tal como sucede en el problema RCPSP, en el cual la duración del proyecto es una medida exacta de la idoneidad del individuo.

Algoritmo 4.1 Algoritmo Genético Híbrido MultiModo (MM-HGA)

```
MM-HGA(Pop_Size,end_cond)
begin
P=Generar_Poblacion_Inicial(Pop_Size)
while NOT (end_cond) do
   P = Seleccion(P)
   P = Cruce(P)
   P = Mutacion(P)
   para todo j en P
      if j es factible con respecto a NR
        j=MM-FBI(j)
        MejorIndividuo=EvaluacionPoblacion(P)
      end if
   end for
end while
```

Aunque el objetivo del MRCPSP también es minimizar la duración del proyecto, en este caso el makespan de la programación asociada a un individuo no es un buen indicador del desempeño. Esto sucede porque individuos con makespan bajo pueden ser no factibles con respecto a los recursos no renovables.

No obstante, queremos mantener dentro de la población estos individuos no factibles. Por lo tanto deben tener también un valor que indique su aptitud, siempre y cuando no sea mayor que el de cualquier individuo factible. Esto quiere decir que un individuo no factible debe ser penalizado pero debe poder ser considerado en el proceso evolutivo.

En la literatura se encuentran dos funciones de penalización diferentes: la de Hartmann [37] y la de Alcaraz et al. [3] que ya han sido presentadas en la sección 2.5.8.

Las desventajas de la función de penalización de Hartmann son dos. Por un lado la cota superior de la duración del proyecto es muy alta, por lo que la penalización de los individuos no factibles es tan fuerte que en términos prácticos no son elegibles para la siguiente generación. De otra parte, el valor de la función objetivo no discrimina de manera eficiente los individuos no factibles. Esta función da la misma aptitud a dos individuos con el mismo exceso de uso de recursos sin importar la duración del proyecto.

La función propuesta por Alcaraz et al. supera estos inconvenientes. Sin embargo, esta función está construida por medio de la adición de unidades de tiempo (makespan del Individuo) y unidades de recursos (del exceso de uso de recursos no renovables). La magnitud de dichos aspectos de la solución pueden distorsionar el sentido de la función de evaluación.

Para superar esta limitación, se propone una nueva función de evaluación donde los dos aspectos de la solución son considerados, pero de una manera normalizada de tal manera que se elimine su orden de magnitud. La nueva función de fitness que se calcula para cada individuo con las expresiones de la ecuación 4.4 que dependen de si un individuo I es factible con respecto a los recursos renovables:

$$f(I) = \begin{cases} 1 - \frac{C_{max}^{pop} - C_{max}(I)}{C_{max}^{pop}} & si L^{\nu}(I) = 0, \\ 1 + \frac{C_{max}(I) - CC_{min}}{C_{max}(I)} + \sum_{k \in R^{v}} \max\left\{0, \frac{\sum_{j=1}^{J} r_{jmk} - a_k}{a_k}\right\} & \text{en otro caso.} \end{cases} \tag{4.4}$$

donde $L^{\nu}(I)$ representa el exceso de uso en los recursos no renovables (definido en la ecuación 2.5.8.2), $C_{max}(I)$ es el makespan del individuo I, C_{max}^{pop} es el makespan máximo de los individuos factibles de la generación actual y CC_{min} es la duración del proyecto utilizando el método del camino crítico calculado con la duración mínima de las actividades.

El individuo factible con el mayor makespan tendrá una aptitud de 1 mientras que el individuo con menor makespan tendrá un valor de aptitud cercano a cero.

Los individuos no factibles siempre tendrán un valor en su función de evaluación mayor a uno, por lo tanto tendrán menos probabilidad de ser seleccionados para el proceso evolutivo. A ello se suma la desviación normalizada del makespan del individuo con respecto al límite inferior calculado con el camino crítico y la suma normalizada del exceso de recursos utilizados. Con ello se pueden superar las desventajas presentadas por las otras funciones de evaluación presentadas en la literatura.

4.2.3.2. Población Inicial

Debido a la complejidad de la asignación de modos factibles, el primer individuo de la población inicial se genera mediante el criterio MNR. Dicho individuo obtiene una asignación de modos que es factible con respecto al uso de recursos no renovables con una gran probabilidad. Para introducir diversidad, la asignación de modos de los demás individuos está basada en una versión aleatorizada del criterio MNR. Para cada individuo se sigue el siguiente procedimiento:

Paso 1 El proceso comienza con la asignación de modos según el criterio MNR
Paso 2 Se cambia el modo de J/2 actividades
Paso 3 Si la asignación de modos es factible termina el procedimiento
En caso contrario las actividades se ordenan de manera aleatoria y cada actividad trata de cambiar su modo a otro que permita reducir el exceso de recursos

Paso 4 Si la asignación de modos es factible se termina el procedimiento. En caso contrario retorne al *Paso 1*

Este proceso se repite hasta un número máximo de iteraciones.

4.2.3.3. Nuevo Operador de Mutación para la lista de modos

Debido a la importancia de mantener tanto la calidad de las soluciones como la diversidad de la población, hemos diseñado un método de mutación específico para la lista de modos.

El nuevo operador de mutación tiene un procedimiento diferente dependiendo de si el individuo es factible o no con respecto a los recursos no renovables:

1. Si el individuo es factible con respecto a los recursos renovables, cada actividad cambia aleatoriamente su modo con la probabilidad p_{muta}
2. Si el individuo no cumple las restricciones en cuanto a los recursos renovables, se aplica un nuevo operador de mutación que ha sido denominado *mutación masiva*, cuyo objetivo es incrementar la diversidad de la asignación de modos. Este operador trabaja de la siguiente manera: escoge de manera aleatoria una actividad de la lista de actividades y le asigna un modo de ejecución (que podría ser el mismo que tenía anteriormente) Este proceso termina cuando la asignación de modos es factible o cuando todas las actividades de la lista han sido consideradas.

La principal ventaja de la *mutación masiva* es que permite mantener la factibilidad de las soluciones a la vez que mantiene la diversidad en la lista de modos de ejecución.

4.3. Evaluación de las aportaciones para el MRCPSP

Esta sección está dedicada a la evaluación de los métodos desarrollados para la solución del problema MRCPSP. Las propuestas realizadas para el problema MRCPSP se integran todas en un algoritmo genético híbrido, del cual se presenta el esquema general en el algoritmo 4.2. Para evaluar su eficiencia resolvemos los problemas de la librería estándar de pruebas PSPLIB para el caso multimodo y los problemas del conjunto de Boctor. Finalmente comparamos su desempeño con los métodos con mejores resultados presentados en la literatura.

Como primer paso, se ejecuta un procedimiento de pre-proceso de Sprecher et al.[89]para acotar el espacio de búsqueda, eliminando los modos ineficientes y no ejecutables, así como los recursos renovables redundantes. En este punto se genera la población inicial y cada individuo es decodificado. Si es posible, se realiza el proceso de búsqueda local para disminuir la duración de la

Algoritmo 4.2 Algoritmo Genético Híbrido MultiModo (MM-HGA)

```
MM-HGA(Pop_Size,end_cond)
begin
P=Generar_Poblacion_Inicial(Pop_Size)
while NOT (end_cond) do
  P = Seleccion(P)
  P = Cruce(P)
  P = Mutacion(P)
  para todo j en P
    if j es factible con respecto a NR
      j=MM-FBI(j)
      MejorIndividuo=EvaluacionPoblacion(P)
    end if
  end for
end while
```

secuencia. La población se evalúa y los operadores genéticos producen la siguiente generación.

Debido a los buenos resultados obtenidos en el capítulo anterior se ha escogido una representación basada en la lista de actividades. Debido a que cada actividad se puede ejecutar en uno de múltiples modos, cada individuo se representa mediante dos listas: una lista de modos y una lista de actividades. Se incluye un gen para indicar si la decodificación se hace mediante el esquema serie o el paralelo, tal como lo hace Alcaraz et al. Se incluye un gen adicional que codifica la dirección de la programación, ya sea forward o backward.

El método de selección utilizado es el de 2-torneo, para el cruce de los individuos se utiliza el de dos puntos con una probabilidad del 90 % y en cuanto a la mutación de la lista de actividades es el método de Boctor con una probabilidad del 5 %.

Con el fin de diseñar un buen AG, se incluye un procedimiento de sustitución aleatoria de la población. Esta operación se aplica a cada generación, con una probabilidad p_{reemp}. Si se lleva a cabo en la actual generación, cada individuo de la población puede ser cambiado con una probabilidad con una probabilidad de p_{exc} por una solución generada mediante el proceso MNR con una iteración en el proceso de factibilización. Este procedimiento permite a la población introducir cierta variabilidad cuando ha convergido prematuramente o si está atrapada en un óptimo local. Los valores dep_{reemp} y p_{exc} se establecen a 0.7 y 0.1, respectivamente.

Para garantizar la supervivencia del más apto se utiliza el elitismo.

A continuación presentamos la evaluación de cada uno de los nuevos métodos propuestos y finalizaremos con los resultados globales del Algoritmo Genético Híbrido.

4.3.1. Desempeño de la Función Evaluación de los individuos

Para probar el desempeño de la nueva función de evaluación se compara su resultado con las funciones propuestas por Hartmann [37] y Alcaraz et al. [3]. Para ello se utiliza el MM-HGA con cada una del as funciones de evaluación y se resuelven las instancias de la librería PSPLIB con 10, 12, 14, 16, 18 y 20 actividades utilizando el mismo esfuerzo computacional (5000 iteraciones). La tabla 4.4 presenta el el porcentaje de desviación sobre la solución óptima (%ADOS) de las tres funciones de desempeño.

Tabla 4.4. %ADLB de la función de evaluación de los individuos

		% ADOS					
Función	Referencia	J10	J12	J14	J16	J18	J20
MM-HGA	[68]	0.06	0.17	0.32	0.44	0.63	0.87
Alcaraz et al.	[3]	0.07	0.16	0.33	0.48	0.67	0.93
Hartmann	[37]	0.10	0.18	0.35	0.51	0.65	0.85

La nueva función de fitness tiene un comportamiento más robusto y por tanto es más apropiada para su uso en el MRCPSP. Concretamente, en 4 de las 6 instancias la nueva función de fitness obtiene los mejores resultados en cuanto al porcentaje de desviación del óptimo.

Adicionalmente, mediante una prueba t sobre todas las instancias se puede demostrar que la nueva función de fitness obtiene mejores resultados que las otras funciones con un nivel de confianza del 90 %

Finalmente, la tabla 4.5 muestra el porcentaje de soluciones óptimas obtenidas por cada una de las funciones evaluadas.

Tabla 4.5. Porcentaje de soluciones óptimas encontradas

		Porcentaje de óptimos					
Función	Referencia	J10	J12	J14	J16	J18	J20
MM-HGA	[68]	98.51	96.53	92.92	90.00	84.96	80.32
Alcaraz et al.	[3]	98.32	96.71	92.20	89.09	83.88	79.24
Hartmann	[37]	97.76	96.34	91.83	88.36	84.24	80.87

4.3.2. Evaluación de la Asignación de Modos y de la Generación de la Población Inicial

El AG comienza con la generación de la población inicial, que es un conjunto de Pop_Size soluciones. Cada individuo es obtenido mediante la generación aleatoria de los genes p/s y b/f y una selección de modo que se describe a continuación. Con esta información, cada solución se obtiene utilizando la heurística basada en la regla de prioridad del mínimo tiempo más tardío de finalización (MinLFT).

Uno de los aportes al MRCPSP es el nuevo método de selección de modos de la población inicial. Este nuevo método se basa en la normalización de los recursos (Sección 4.2.2). Para evaluar su efectividad se compara con los dos métodos presentados en la literatura: la selección aleatoria de modos (RND) y el modo con máxima duración (MaxDur). Este último método selecciona el modo con la duración más larga, que por lo general es la que menos recursos consume.

La tabla 4.6 muestra los resultados obtenidos.

Tabla 4.6. Porcentaje de Asignación de Modo Factible según el criterio MNR

Método	J10	J12	J14	J16	J18	J20	J30
RND	53.92	56.12	55.17	54.91	55.80	58.12	59.78
MaxDur	90.30	93.42	91.47	93.27	93.38	94.04	95.29
MNR	96.08	95.61	97.46	97.27	98.73	97.83	98.55

El proceso MNR obtiene soluciones factibles en un rango que varía del 95.61 % al 98.73 %, mientras que el método de selección MaxDur no supera el 95.29 % de los casos. Finalmente, la selección aleatoria de modos es la que peor desempeño tiene y no llega nunca al 60 % de asignaciones factibles.

Para generar la población inicial siguiendo el método presentado en la Sección 4.2.2, se define en 200 el número de iteraciones de factibilización del individuo.

Puede parecer que este procedimiento para obtener una asignación de modos factible conlleva un gran tiempo de cómputo. sin embargo, en la práctica al menos un 79 % de individuos se generan en la primera iteración, tal como se observa en la tabla 4.7

4.3.3. Desempeño del operador de mutación masiva en el Algoritmo Genético Básico

El primer análisis que hemos realizado es el estudio de la influencia del operador de mutación masiva descrito en la sección 4.2.3.3. La figura 4.2

Tabla 4.7. Número de iteraciones máximas necesarias para lograr un individuo factible

Iteraciones	J10	J12	J14	J16	J18	J20	J30
0	79.71	82.95	82.64	83.84	85.03	84.61	86.69
1	11.90	10.59	10.00	9.54	8.47	8.38	6.86
2	3.55	3.00	2.99	2.72	2.57	2.52	2.21
3	1.69	1.20	1.27	1.32	1.34	1.36	1.35
4-199	3.14	1.92	2.67	2.13	2.16	2.57	2.57
200	0.00	0.33	0.43	0.45	0.42	0.55	0.32

ilustra los resultados para los conjuntos J10 y J20 en el AG básico, es decir, el MM-FBI método no se aplica.

	J10				J20			
	AG Básico sin Mutación Masiva		AG Básico		AG Básico sin Mutación Masiva		AG Básico	
N Sol	% ADLB	Tiempo CPU	% ADLB	Tiempo CPU	% ADLB	Tiempo CPU	% ADLB	Tiempo CPU
500	4.16	0.01	3.27	0.01	12.26	0.02	11.44	0.02
1000	1.81	0.02	1.17	0.02	7.56	0.03	6.73	0.04
2000	1.07	0.03	0.50	0.03	4.94	0.06	4.34	0.07
3000	0.77	0.05	0.29	0.05	3.95	0.09	3.37	0.10
4000	0.63	0.06	0.22	0.06	3.44	0.12	2.94	0.12
5000	0.61	0.08	0.16	0.08	3.07	0.15	2.61	0.15
6000	0.49	0.09	0.16	0.09	2.77	0.17	2.46	0.18

Figura 4.2. Impacto del operador de mutación masiva en el AG básico

La tabla 4.8 muestra como para los dos conjuntos de problemas, la aplicación de este operador permite la obtención de mejores resultados sin importar el número de programaciones generadas, por ello se incluye en el MM-HGA.

4.3.4. Impacto del método de mejora sobre el Algoritmo Genético Básico

La influencia del método de mejora MM-FBI presentado en la sección 4.2.1 se analiza mediante la resolución de los J10 y J20 con el AG Básico utilizando un esfuerzo computacional que varía de 500 a 6000 soluciones generadas. La figura ilustra los resultados utilizando y sin utilizar el método de búsqueda local.

La utilización del método MM-BFI resulta decisivo por mejorar drásticamente el desempeño del AG. Cuando se considera el conjunto se J10,

Tabla 4.8. Impacto del operador de Mutación Masiva

	J10				J20			
	AG Básico sin Mutación Masiva		AG Básico		AG Básico sin Mutación Masiva		AG Básico	
N Sol	% ADLB	Tiempo CPU	% ADLB	Tiempo CPU	% ADLB	Tiempo CPU	% ADLB	Tiempo CPU
500	4.16	0.01	3.27	0.01	12.26	0.02	11.44	0.02
1000	1.81	0.02	1.17	0.02	7.56	0.03	6.73	0.04
2000	1.07	0.03	0.50	0.03	4.94	0.06	4.34	0.07
3000	0.77	0.05	0.29	0.05	3.95	0.09	3.37	0.10
4000	0.63	0.06	0.22	0.06	3.44	0.12	2.94	0.12
5000	0.61	0.08	0.16	0.08	3.07	0.15	2.61	0.15
6000	0.49	0.09	0.16	0.09	2.77	0.17	2.46	0.18

el AG sin utilizar el método MM-FBI presenta una desviación con respecto a la solución óptima (%ADOS) que varía entre 3,27 % a 0,16 % en función del esfuerzo computacional. (Tabla 4.9)

Tabla 4.9. Desempeño del AG Básico y del MM-HGA en los conjuntos J10 y J20

	J10				J20			
	AG Básico		MM-HGA		AG Básico		MM-HGA	
N Sol	% ADLB	Tiempo CPU	% ADLB	Tiempo CPU	% ADLB	Tiempo CPU	% ADLB	Tiempo CPU
500	3.27	0.01	0.59	0.01	11.44	0.02	2.88	0.02
1000	1.17	0.02	0.26	0.02	6.73	0.04	2.03	0.04
2000	0.50	0.03	0.14	0.03	4.34	0.07	1.36	0.06
3000	0.29	0.05	0.08	0.04	3.37	0.10	1.08	0.08
4000	0.22	0.06	0.08	0.06	2.94	0.12	0.95	0.11
5000	0.16	0.08	0.06	0.07	2.61	0.15	0.87	0.13
6000	0.16	0.09	0.04	0.10	2.46	0.18	0.81	0.18

Sin embargo, cuando se utiliza el MM-BFI, la variación sobre la solución óptima cae hasta un 0,59 % como máximo y un mínimo de 0,04 %. Para el caso J120 las conclusiones son similares (Figura 4.3 y Tabla 4.9)

4.3.5. Desempeño del MM-HGA en el problema MRCPSP

Con el fin de probar el rendimiento del MM-HGA propuesto, se utilizan los conjuntos J10, J12, J14, J16, J18, J20 y J30 de la librería de pruebas PSPLIB. Cada conjunto se resuelve utilizando un esfuerzo computacional de 1000, 3000 y 5000 soluciones generadas.

La tabla 4.10 muestra para cada conjunto el promedio de la desviación con respecto a la cota inferior (%ADLB), la desviación estándar del %ADLB, el

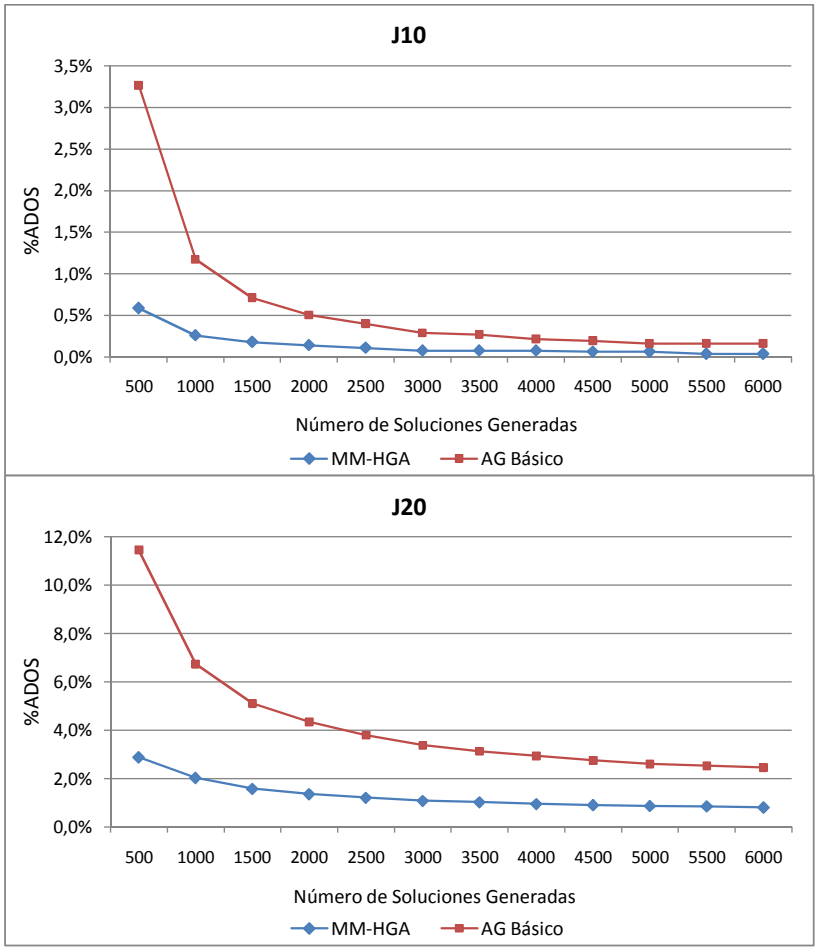


Figura 4.3. Desempeño del AG Básico y del MM-HGA en los conjuntos J10 y J20

porcentaje de soluciones óptimas encontradas y el tiempo de CPU requerido en segundos.

Tal como se describe en la sección 2.6.1.2 las soluciones óptimas son conocidas para los conjuntos de 10 a 20 actividades y para el conjunto de 30 actividades dicha cota es la duración del camino crítico cuando se utiliza el modo con menor duración para cada actividad.

Los problemas utilizados en esta primera evaluación pueden ser considerados como proyectos de tamaño pequeño. Por lo tanto, a fin de evaluar el desempeño del MM-HGA, con problemas de tamaño medio y grande hemos diseñado dos nuevos conjuntos de 60 y 120 actividades. Estos problemas pueden descargarse de http://users.dsic.upv.es/grupos/gps.

Tabla 4.10. Desempeño del MM-HGA con un esfuerzo computacional de 1000, 3000 y 5000 soluciones generadas

No. Soluciones	%ADLB	Max.	Desv. Est.	% Óptimos	Tiempo CPU(s)
J10					
1000	0.26	16.67	1.31	95.34	0.02
3000	0.08	6.25	0.59	98.13	0.05
5000	0.06	6.25	0.55	98.51	0.08
J12					
1000	0.75	17.65	2.24	87.02	0.02
3000	0.31	17.65	1.40	93.97	0.06
5000	0.17	10.00	0.97	96.53	0.10
J14					
1000	0.99	15.79	2.36	81.13	0.03
3000	0.43	8.33	1.38	90.38	0.07
5000	0.32	8.33	1.22	92.92	0.11
J16					
1000	1.29	14.29	2.47	74.55	0.03
3000	0.63	9.68	1.70	86.18	0.08
5000	0.44	9.68	1.39	90.00	0.12
J18					
1000	1.55	18.18	2.91	71.74	0.03
3000	0.77	12.50	1.85	82.25	0.08
5000	0.63	11.11	1.63	84.96	0.13
J20					
1000	2.03	17.65	3.28	64.80	0.04
3000	1.08	14.71	2.26	76.71	0.10
5000	0.87	10.71	1.97	80.32	0.15
J30					
1000	16.56	150.00	26.90	49.10	0.06
3000	15.30	142.31	25.69	51.62	0.13
5000	14.77	138.46	25.08	52.35	0.21

La tabla 4.11 presenta los resultados obtenidos para estos problemas. Se se concluye que a pesar del incremento del tamaño, el algoritmo obtiene buenos resultados y es eficiente computacionalmente.

Tabla 4.11. Desempeño del MM-HGA con un esfuerzo computacional de 1000, 3000 y 5000 soluciones generadas - Problemas de 60 y 120 actividades

	J60				J120			
No. Soluciones	% ADLB	Max	Desv Est	Tiempo CPU (sg)	% ADLB	Max	Desv Est	Tiempo CPU (sg)
1000	16.44	156.67	26.77	0.12	16.34	154.00	26.42	0.41
3000	14.68	156.67	25.08	0.27	14.32	148.00	24.44	0.91
5000	14.10	143.33	24.45	0.42	13.65	142.00	23.71	1.40

4.4. Comparación con los mejores algoritmos reportados

Para comparar el desempeño del nuevo HGA con los mejores algoritmos publicados en la literatura hemos obtenido la solución para los problemas del conjunto J10 con el algoritmo MM-HGA propuesto considerando un esfuerzo computacional desde 1000 a 6000 soluciones generadas. Los resultados se presentan en la tabla 4.12.

Es interesante señalar que el MM-HGA está entre los mejores algoritmos publicados hasta el momento y supera a muchos de ellos con la mitad del esfuerzo computacional. El algoritmo mantiene la tasa de mejora a medida que se generan más soluciones y que está cerca de ser óptimo cuando el esfuerzo de cálculo es de 6000 soluciones generadas. El MM-HGA obtiene una media de desviación con respecto a la óptima de 0,04 %.

Tabla 4.12. Comparación con otros métodos heurísticos para el conjunto J10

Autores	Referencia	% ADOS	% Soluciones óptimas	Soluciones Generadas
Van Petghem & Vanhoucke	[104]	0.01	99.63	5000
MM-HGA	[68]	0.04	99.07	6000
MM-HGA	[68]	0.06	98.51	5000
MM-HGA	[68]	0.08	98.13	4000
MM-HGA	[68]	0.08	98.13	3000
Hartman	[37]	0.10	98.10	6000
Zhang	[106]	0.11	97.90	*
MM-HGA	[68]	0.14	96.83	2000
Ranjbar et al.	[82]	0.18	*	5000
Alcaraz et al.	[3]	0.19	96.50	6000
Bouleimen & Lecoq	[15]	0.21	96.30	*
MM-HGA	[68]	0.26	95.34	1000
Kolisch & Drexl	[55]	0.50	91.80	6000
Özdamar	[75]	0.86	88.10	6000

Adicionalmente, se lleva a cabo un análisis más detallado teniendo en cuenta la totalidad de los conjuntos de la librería. La comparación se hace con respecto a los datos publicados en la literatura y para todos ellos el esfuerzo computacional corresponde a 5000 soluciones generados. Los resultados se presentan en la tabla 4.13 y evidencian que el algoritmo propuesto es eficiente independientemente del conjunto de casos utilizados.

Finalmente, el algoritmo se prueba con los problemas de Boctor [12]. Son problemas de tamaño mediano (50 actividades) y grande (100 actividades) cuya principal característica es el hecho de que sólo tienen en cuenta recursos renovables. Estos problems se describen en la el Apéndice 2.6.1.2

Tabla 4.13. Comparación con otras heurísticas sobre el %ADLB y 5000 soluciones generadas

Autores	Referencia	J10	J12	J14	J16	18	J20
Van Petghem & Vanhoucke	[104]	0.01	0.09	0.22	0.32	0.42	0.57
MM-HGA	[68]	0.06	0.17	0.32	0.44	0.63	0.87
Hartman	[37]	0.06	0.14	0.44	0.59	0.99	1.21
Zhang	[106]	0.11	0.17	0.41	0.83	1.33	1.79
Ranjbar et al.	[82]	0.18	0.65	0.89	0.95	1.21	1.64
Alcaraz et al.	[3]	0.24	0.73	1.00	1.12	1.13	1.91
Jozefowska et al.	[46]	1.16	1.73	2.60	4.07	5.52	6.74

El desempeño de la MM-HGA se compara con el AG propuesto por Alcaraz et al [3] y por Van Peteghem & Vanhoucke [104]. (Tabla 4.14).

Tabla 4.14. Resultados comparativos para los conjuntos de Boctor

Autores	Referencia	B octor 50		B octor 100	
		1000	5000	1000	5000
Van Peteghem & Vanhoucke	[104]	27.36	23.41	29.70	24.67
MM-HGA	[68]	24.89	23.70	26.96	24.85
Alcaraz et al.	[3]	33.83	26.52	51.85	59.16

4.5. Conclusiones

En este capítulo hemos presentado las diferentes aportaciones para la solución del MRCPSP. Uno de los aspectos más importantes que se abordan en el problema MRCPSP es la extensión del método de mejora FBI al caso de múltiples modos. Este método de mejora aprovecha la posibilidad de cambiar el modo de ejecución de las actividades para reducir la duración del proyecto sin sobrepasar la capacidad de los recursos disponibles.

La asignación de modos en el MRCPSP es un problema NP-completo, sin embargo el método que propuesto (MNR) genera en todos los conjuntos más del 95 % de soluciones factibles sin que implique un mayor esfuerzo computacional.

Estas dos propuestas se integran en un algoritmo genético híbrido que aprovecha el método MNR para generar una población inicial de alta calidad sin que ello implique un esfuerzo computacional adicional. El nuevo operador de mutación masiva permite la factibilización de soluciones a la vez que introduce diversidad en la población ayudando a la exploración del espacio de búsqueda. Finalmente, La nueva función de evaluación de los individuos

supera las desventajas de las funciones presentadas con anterioridad en la literatura y guía la evolución del algoritmo genético hacia regiones promisorias del espacio de solución.

El algoritmo genético híbrido propuesto se destaca dentro de los algoritmos publicados en la literatura, pues resuelve de manera eficiente problemas de tamaño pequeños, medianos y grandes tanto en los conjuntos estándar de pruebas (PSPLIB y Boctor) como en los dos conjuntos generados específicamente para este caso.

5

Generación y Evaluación de Programaciones Robustas

5.1. Introducción

En el ámbito de los problemas de programación de proyectos pueden surgir diferentes fuentes de incertidumbre: la duración de las actividades puede ser diferente de la planeada inicialmente, los recursos pueden tener menor capacidad de lo esperado (por ejemplo fallos en la maquinaria), dejando inutilizable la programación construida con los datos determinísticos de la duración de las actividades o la disponibilidad de recursos. Por lo tanto, la programación de proyectos debe ser un proceso continuo que permita responder a las situaciones inesperadas y al entorno variable. En este contexto, la generación de programaciones robustas desempeña un papel crucial.

En este capítulo hacemos la propuesta de un algoritmo genético que inserta tiempos de seguridad para construir una programación robusta. El algoritmo tiene una función de evaluación de los individuos basada en el impacto de los buffers en la estabilidad de la secuencia. Debido a las dificultades de comparación con lo métodos de otros autores, desarrollamos un método aleatorio de generación de programaciones robustas. Para la evaluación de la robustez, las programaciones generadas se someten a un proceso de simulación con dos escenarios de variabilidad. Así mismo, proponemos dos medidas de la robustez de las secuencias una vez ejecutadas.

Los métodos desarrollados se implementaron en C y se compilaron con Microsoft Visual C++ v.6.0 de Microsoft. Para la evaluación se utilizó un ordenador bajo Windows XP con un procesador Intel Core duo de 3GHz y 1.99 GB en RAM.

Este capítulo se estructura de la siguiente manera: la sección 5.2 está dedicada al algoritmo genético propuesto como método proactivo de generación de programaciones robustas. Su evaluación y comparación se hace en la sección 5.3. Finalmente, las conclusiones de este capítulo se detallan en la sección 5.4.

5.2. Aportaciones en la generación proactiva de programaciones

En el problema de generación de secuencias robustas las aportaciones están centradas en el desarrollo de un enfoque basado en algoritmos genéticos como método proactivo. Se diseña una función de fitness específica para el problema, la cual tiene un factor asociado a la importancia de la actividad dentro de la robustez de la secuencia.

Para medir la efectividad del método propuesto, la secuencia obtenida se somete a un proceso de simulación y se evalúa su robustez, para ello hemos diseñado dos medidas que permiten evaluarla.

5.2.1. Algoritmo Genético para generación de programaciones robustas - RGA

En este problema, las actividades están relacionadas tanto por relaciones de precedencia como por el uso de recursos, por tanto cualquier pequeño retraso (retraso primario) puede tener efectos imprevistos en las actividades sucesivas (retraso secundario). Se pueden programar secuencias más tolerantes a las perturbaciones estocásticas insertando mayores tiempos de seguridad para absorber los retrasos.

Sin embargo, la inserción de dichos tiempos implica un menor uso de recursos con lo que se incrementan los costos y puede significar una pérdida de la satisfacción del cliente por tener que esperar más de lo necesario. El objetivo es lograr un balance entre los dos aspectos.

Siguiendo una práctica habitual en la programación de proyectos (Herroelen & Leus [43], [44], Van de Vonder et al. [101]) adoptamos un enfoque de dos etapas donde la primera resuelve el problema del RCPSP y luego en la segunda etapa, se adicionan tiempos de seguridad (buffers) a la secuencia inicial.

La construcción de secuencias robustas puede hacerse descomponiendo el problema, encontrando primero el orden ideal de las actividades y luego adicionando los tiempos de seguridad tal como Van de Vonder et al. [101] o integrando los dos procedimientos como Lambrecths et al. [62].

Hemos escogido el segundo enfoque. Cualquier solución factible para el RCPSP utilizando la duración media de las actividades puede servir como dicha secuencia inicial. Luego, se permite a las actividades cambiar su posición en la lista a la vez que se introducen los tiempos de seguridad. El objetivo del algoritmo propuesto es determinar el orden de ejecución de las actividades, así como seleccionar las actividades a las que se les insertarán los buffers y el tamaño del mismo.

En el algoritmo 5.1 se presenta el funcionamiento general del método propuesto para la generación proactiva de programaciones robustas. Las principales características del algoritmo se detallan a lo largo de las siguientes secciones.

Algoritmo 5.1 Algoritmo Genético Programaciones Robustas (RGA)

```
RGA(Pop_Size,end_cond)
begin
P=Generar_Poblacion_Inicial(Pop_Size)
while NOT (end_cond) do
   P = Seleccion(P)
   P = Cruce(P)
   P = Mutacion(P)
   MejorIndividuo=EvaluacionPoblacion(P)
end while
```

5.2.1.1. Representación de los Individuos

En este trabajo utilizamos una representación basada en la lista de actividades (AL). Cada individuo está representado por una doble lista: la lista de actividades y una lista de buffers, cada una de las cuales tiene tantas posiciones como actividades hay en el proyecto.

La lista de actividades es factible con respecto a las relaciones de precedencia, es decir cada actividad en la lista está después de todos sus predecesores. La lista de buffers indica el número de unidades de tiempo en que debe incrementarse el tiempo de finalización determinado por el esquema de generación serie. Esta representación ha sido utilizada por Lambrechts et al.[62]. La codificación de la representación se ilustra en la figura 5.1

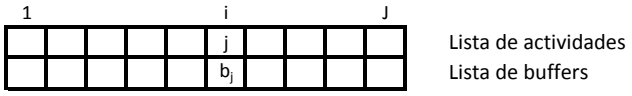


Figura 5.1. Representación del Individuo RGA

5.2.1.2. Generación de la Población Inicial

La población inicial debe satisfacer dos condiciones para asegurar el desempeño del AG: diversidad y calidad. La población debe estar compuesta por individuos diversos para así permitir al algoritmo explorar diversas zonas del espacio de solución desde sus etapas tempranas. Adicionalmente, la calidad de los individuos debe permanecer aceptable pues la calidad de la población es el principal criterio para la evolución.

Para mantener la calidad, el primer individuo de la población es el que se obtiene con el algoritmo genético de población dinámica y sustitución inmediata que se presentó en la sección 3.4.2. Para ganar en diversidad, los otros individuos se construyen cambiando una actividad en la AL utilizando el operador de inserción de Boctor presentado en la sección . Este operador

mantiene la precedencia en la lista de actividades. La lista de buffers para todos los individuos se inicializa en cero. El tamaño de la población es de Pop_Size individuos.

5.2.1.3. Operadores Genéticos

El procedimiento comienza con la generación de la población inicial y la evaluación de cada uno de los individuos. Se utiliza el método de de 2Torneo para la selección de los padres. Estos individuos se recombinan mediante el cruce de 2 puntos descrito en la sección 2.5.4, con una probabilidad P_{cruce} del 90 %.

El operador de mutación se aplica a los dos componentes de cada individuo, es decir, lista de actividades y la lista de buffers. En el primer caso, el procedimiento de mutación utilizado es el operador de inserción de Boctor presentado en la sección 5.2.1.2. Hemos utilizado una probabilidad de mutación P_{mut_AL} de 0.05.

Por otra parte, el operador de mutación en la lista de buffers aumenta o disminuye en una unidad el valor actual del buffer actual asignado. La probabilidad utilizada es P_{mut_BL}=0.2, que se divide en partes iguales para los incrementos y decrementos. Sin embargo, como el valor de un buffer no puede ser nunca negativo un valor de cero nunca puede ser disminuido. Los individuos así generados se evalúan. Cada iteración corresponde a una nueva generación.

5.2.1.4. Evaluación de los Individuos

En el caso de la robustez a definición de la función de fitness no es sencilla, ya que la función de evaluación no coincide con la función objetivo del problema. La evaluación analítica de la robustez de una secuencia es complicada. Existen dos enfoques para su medición.

Algunos autores lo hacen mediante simulación, sin embargo requiere demasiado tiempo para evaluar cada uno de los individuos de la población, convirtiéndolo en ineficaz. El otro enfoque es el diseño de medidas indirectas que permitan hacer una buena estimación de la magnitud de la estabilidad de la secuencia. Hemos escogido este segundo enfoque para nuestro algoritmo.

El primer paso es medir el impacto de los buffers en la estabilidad de la programación. Para ello, utilizamos la medida propuesta por Lambrecths et al. [61] basada en la holgura libre con recursos o *Resource Free Slack*(RFS) de cada actividad:

$$\sum_{k=1}^{RFS} e^{-k} \tag{5.1}$$

La RFS tiene en cuenta no sólo las relaciones de precedencia, sino también los recursos disponibles.

En segunda instancia, es necesario medir el impacto del retraso de las actividades en la red. Para ello utilizamos el *peso de inestabilidad* de una actividad (w_i), el cual puede ser visto como la contribución de la RSF de la actividad i en la función objetivo [62].

De otra parte, es necesario tener en cuenta que es posible que se en la evolución del algoritmo se generen soluciones no factibles, es decir que el proyecto no cumpla el plazo que se le ha establecido. Estos individuos deben tener también un valor que indique su aptitud pero debe ser penalizado para que siempre sea menor que el de cualquier individuo factible.

Basándonos en las anteriores consideraciones proponemos la siguiente función de fitness dependiendo de si un individuo es factible ($s_n \leq \delta_n$) o no lo es:

$$f(I) = \begin{cases} \sum_{j=1}^{n} w_j \sum_{k=1}^{RFS} e^{-k} & \text{si } s_n \leq \delta_n \\ \sum_{j=1}^{n} w_j \sum_{k=1}^{RFS} e^{-k} \div (s_n - \delta_n + 1) & \text{en otro caso.} \end{cases} \tag{5.2}$$

Peso de Inestabilidad de las Actividades

Como se mencionó en la sección anterior, los pesos de las actividades pueden ser utilizados para medir el impacto del retraso de una actividad en la estabilidad de la secuencia. En los trabajos previos como Lambrechts et al. [61], y Van de Vonder et al[102], [103], [101] se asume que los pesos de las actividades representan el costo de retrasar la actividad. Sin embargo son generados de manera aleatoria.

Debido a la importancia de los pesos de inestabilidad en la función objetivo, tal como se presentó en la Sección 2.2.3 queremos valorar la posibilidad de que alguna de las características de las actividades pudieran ser determinantes para la estabilidad de la secuencia. También valoramos el escenario donde todas las actividades tienen el mismo impacto en la estabilidad y todos los pesos son iguales entre si e iguales a 1.

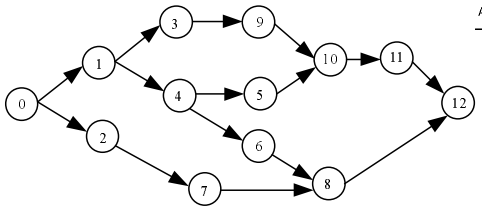
En la tabla 5.1 presenta los diferentes conjuntos de pesos que se proponen para su valoración.

Tabla 5.1. Pesos de las actividades

Descripción	Nombre	Valor
Pesos Unitarios	Unit	$w_i = 1$
Número de Sucesores Inmediatos	InmSuc	$w_i = \sum_{j \in IS_i}$
Número de Sucesores Totales	TotSuc	$w_i = \sum_{j \in TS_i}$
Número de Predecesores Inmediatos	InmPred	$w_i = \sum_{j \in IP_i}$
Número de Predecesores Totales	TotPred	$w_i = \sum_{j \in TP_i}$
Duración de la actividad	Dur	$w_i = d_i$

5.2.1.5. Ejemplo

Para ilustrar lo descrito anteriormente, utilizamos la instancia representada en la figura 5.2. Las actividades 0 y 12 son las actividades ficticias de inicio y fin del proyecto, con duración y utilización de recursos igual a cero.



Actividad	Actividades Sucesoras Inmediatas	Duración (dias)	Consumo de Recursos R1	R2
1	3.4	5	3	2
2	7	5	2	4
3	8	4	3	1
4	5.6	2	4	3
5	9	3	2	0
6	10	3	1	1
7	10	4	3	1
8	9	3	2	2
9	11	3	3	2
10	--	2	4	1
11	--	3	5	4

Figura 5.2. Instancia de Ejemplo Disponibilidad (R1, R2)=(6,5)

El primer paso es resolver el problema RCPSP obteniendo un makespan de 22 y un plazo de 29 unidades de tiempo. Esta solución (Figura 5.3) es el punto de partida del AG. Como se observa, las unidades de tiempo adicionales se incluyen al final del proyecto.

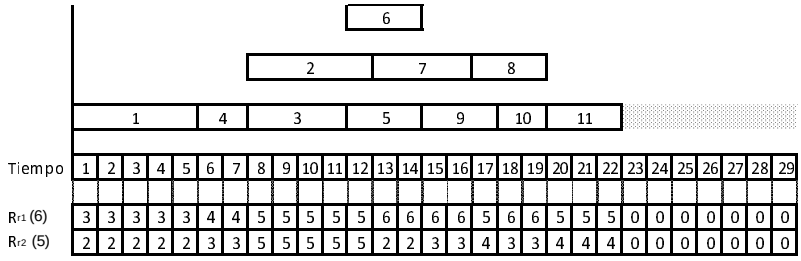


Figura 5.3. Duración Mínima del Proyecto

La Figura 5.4 ilustra el resultado del AG luego de evaluar 1000 individuos utilizando la función de fitness de la ecuación 5.2, y la duración de las actividades como peso de inestabilidad (w_i).

La tabla 5.2 muestra la manera de calcular el fitness para el individuo de la figura 5.4. Para calcular la RFS se determina el número de unidades de tiempo que se puede retrasar una actividad sin que se retrasen las restantes, ya sea por relaciones de precedencia o por utilización de recursos. La última columna se obtiene al multiplicar el peso de las actividades, en este caso su duración, con la función modificada de la holgura. La suma de todos los factores de la última columna es el valor de aptitud del individuo.

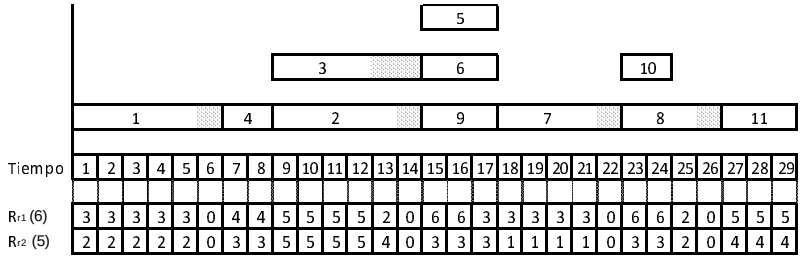


Figura 5.4. Programación Robusta

Tabla 5.2. Cálculo de la función de fitness

Actividad	Buffer	s_i	d_i	f_i	RFS	$e^{-\mathrm{RFS}}$	$w_i \times e^{-\mathrm{RFS}}$
0	0	0	0	0	0	0.000	0.000
1	1	0	5	5	1	0.368	1.839
4	0	6	2	8	0	0.000	0.000
3	2	8	4	12	2	0.503	2.013
2	1	8	5	13	1	0.368	1.839
5	0	14	3	17	5	0.578	1.734
6	0	14	3	17	5	0.578	1.734
9	0	14	3	17	5	0.578	1.734
7	1	17	4	21	1	0.368	1.472
8	1	22	3	25	1	0.368	1.104
10	0	22	2	24	2	0.503	1.006
11	0	26	3	29	0	0.000	0.000
12	0	29	0	29	0	0.000	0.000
					Fitness		**14.476**

5.2.2. Medidas de Robustez de la Secuencia Realizada

Con el objetivo de evaluar la robustez de las soluciones generadas por el AG hemos tenido en cuenta que hay dos factores relevantes. Por una parte es importante la estabilidad de la secuencia, entendida como el cumplimiento de los tiempos de inicio de las actividades. Adicionalmente, desde el punto de vista del gerente del proyecto es importante cumplir con el plazo de finalización del mismo.

Por ello hemos propuesto dos medidas de robustez diferentes para cada una de las secuencias generadas. Para el objetivo de la estabilidad de la programación hemos definido la **Robustez Promedio de la Secuencia Base** (ABSR) como la suma de la desviación absoluta entre el tiempo de inicio planeado (s_i) y el tiempo de inicio realizado ($\mathbf{s}_i$)dividido por el número total de actividades.

$$\text{ABSR} = \frac{\sum_{j=1}^{N} w_j |s_j - \mathbf{s_j}|}{N} \tag{5.3}$$

En cuanto a la robustez de la finalización del proyecto, definimos **Promedio Porcentual de Retraso de la Finalización del Proyecto** (%ADPC):

$$\%\text{ADPC} = \frac{s_n - \mathbf{s}_n}{s_n} \times 100 \tag{5.4}$$

Las secuencias generadas por el Algoritmo Genético y sometidas al proceso de simulación se evalúan mediante estas dos medidas.

5.3. Evaluación y Comparación

El objetivo de los métodos proactivos es la construcción de una secuencia base robusta, es decir que esté tan protegidas como sea posibles frente a las eventualidades que puedan surgir durante la ejecución del proyecto. Se da por supuesto que es posible que una secuencia robusta, a pesar de estar protegida, se convierta en no factible durante la ejecución del proyecto. Por lo tanto, para hacer la evaluación son necesarios dos pasos adicionales: el diseño de un método reactivo y establecer los escenarios de valoración.

5.3.1. Procedimiento Reactivo

El procedimiento reactivo es aquel que permite reparar la secuencia base generada por el método proactivo cuando ésta no pueda absorber las eventualidades que surgen durante la ejecución y por tanto las actividades deban reprogramarse.

El método que hemos implementado es el siguiente. La secuencia base se ejecuta hasta la finalización de la actividad ficticia fin o hasta que una actividad necesite un tiempo mayor del planeado para terminar su ejecución. Se asume que las actividades no se pueden interrumpir y que una actividad no puede comenzar su ejecución antes de su inicio planeado (*railway scheduling*).

En primer lugar, se construye una lista de actividades que permite programar las actividades según el orden dictado por la secuencia base. Ésta es ordenada crecientemente por el tiempo de inicio de las actividades. Los empates se rompen en orden decreciente del peso de inestabilidad. Si persistiera se escoge la actividad con menor identificador.

La secuencia realizada se construye aplicando el SGS paralelo utilizando la duración actual de las actividades. El proceso empieza cuando ocurre un imprevisto que no puede ser absorbido por la secuencia base; en ese punto se recorre la lista de actividades para seleccionar actividades que aún no han sido programadas. En cada punto de decisión se tratan de programar la mayor cantidad de actividades cumpliendo siempre las restricciones de precedencia y recursos.

5.3.2. Escenarios de Variabilidad

Para simular la duración de las actividades, utilizamos una distribución beta con asimetría a la derecha, con parámetros 2 y 5. El valor de la duración media se corresponde con la duración determinística de la actividad utilizada para construir la secuencia base. Así mismo, asumimos que la variabilidad depende de la actividad y utilizamos dos niveles diferentes.[101]

Alta variabilidad de la duración significa que la duración de la actividad se obtiene de la distribución beta $\beta(2,5)$, con un valor mínimo de $0.25\times E(\mathbf{d}_i)$, la media es $E(\mathbf{d}_i)$ y el valor máximo de la duración corresponde a $2.875\times E(\mathbf{d}_i)$.

De otra parte, el escenario de *baja variabilidad de la duración* significa que el valor de la duración de cada actividad se obtiene de la distribución beta $\beta(2,5)$ y la media es $E(\mathbf{d}_i)$. Sin embargo, el valor mínimo es de $0.75\times E(\mathbf{d}_i)$, y el valor máximo $1.625\times E(\mathbf{d}_i)$.

Para cada instancia se ejecutan 100 simulaciones, en cada una de ellas se genera un conjunto de duración de las actividades según el escenario de variabilidad que se esté evaluando.

El procedimiento reactivo se ejecuta utilizando estas duraciones simuladas de las actividades. Cuando se construye la secuencia realizada, asumimos que en cualquier instante de tiempo únicamente se conocen las duraciones *reales* de las actividades que ya han finalizado en dicho instante.

5.3.3. Método Aleatorio de Inserción de Buffers

A diferencia de otros problemas en el área de programación de proyectos como RCPSP y MRCPSP, no existe una manera estándar de comparar los resultados en el problema de la generación de programaciones robustas. Esto se debe a que los pesos utilizados y la duración de las actividades en el simulador son aleatorios.

Por lo tanto, es necesario implementar un procedimiento que permita evaluar la efectividad del GA diseñado. Para ello hemos desarrollado un procedimiento que de manera aleatoria decide la posición y el tamaño de los buffers. El procedimiento parte de la misma solución inicial y plazo límite para el proyecto que el AG.

El primer individuo es el obtenido como solución del RCPSP. Los otros individuos se generan de la siguiente manera: se genera una lista de actividades de manera aleatoria pero considerando las relaciones de precedencia. Se decodifica con el SGS serie forward. Si su makespan es menor que la fecha límite del proyecto se procede a insertar los buffers, de lo contrario se genera una nueva lista.

La inserción de los buffers funciona de la siguiente manera: se escoge aleatoriamente una actividad y su buffer se incrementa en una unidad. Por medio del SGS serie forward se calcula el makespan. El proceso se detiene cuando la programación excede el plazo del proyecto. La última programación factible obtenida es la que se guarda y es evaluada utilizando la ecuación 5.2.

De esta manera se generan 1.000 individuos y aquel con el mejor valor se considera la solución del procedimiento.

5.3.4. Pesos de Inestabilidad de las Actividades

Con el fin de hacer una valoración más completa de los pesos de inestabilidad, incluimos la propuesta de Lambrechts et al. [61]. En su trabajo se asume que si una actividad j se pospone una unidad de tiempo, todos sus sucesores lo harán. Debido a ello, el peso de inestabilidad de una actividad es la suma de los pesos de inestabilidad de todos sus sucesores:

$$CIW_j = \sum_{i \in TS_j} w_i \tag{5.5}$$

Teniendo en cuenta las consideraciones anteriores, evaluamos dos funciones de fitness diferentes: una utiliza el peso de la actividad de la tabla 5.1 como peso de inestabilidad w_i . La segunda utiliza el peso acumulado w_i=CIW.

Para diferenciar su uso en los procedimientos los hemos identificado de la siguiente manera. El algoritmo genético y el procedimiento aleatorio cuando utilizan el peso de la actividad w_i se denominan GA_w y RND_w. Cuando utilizan el peso acumulado, GA_CIW y RND_CIW.

5.3.5. Robustez Promedio de la Secuencia Base (ABSR)

El primer análisis es el impacto en la métrica ABSR. La figura 5.5 muestra los resultados para los conjuntos de 30, 60 y 120 actividades de la librería PSPLIB en el escenario de alta variabilidad. En todos los casos considerados el AG. En los conjuntos J60 y J120 el AG tiene un desempeño superior sin importar el conjunto de pesos utilizados.

Es notable la variación del conjunto de pesos derivados de los Predecesores Totales (Tabla 5.3). Cuando se utiliza CIW el desempeño se encuentra entre los mejores; sin embargo cuando se utilizan como peso w_i. La razón es el impacto del peso de la actividad ficticia fin, tal como establecieron Van de Vonder et al.[102].

Resultados similares se obtienen para el caso de baja variabilidad (Figura 5.6). El procedimiento AG_w que utiliza los pesos tomados de la duración de las actividades obtiene los mejores resultados en los tres conjuntos (Tabla 5.3).En este escenario, los dos conjuntos de pesos (w y CIW) tienen un desempeño similar, en particular en el conjunto J120.

5.3.6. Promedio Porcentual de Retraso de la Finalización del Proyecto (%ADPC)

En el escenario de alta variabilidad, se revela como crucial utilizar el AG con el conjunto de pesos derivados del número de Predecesores Totales y la

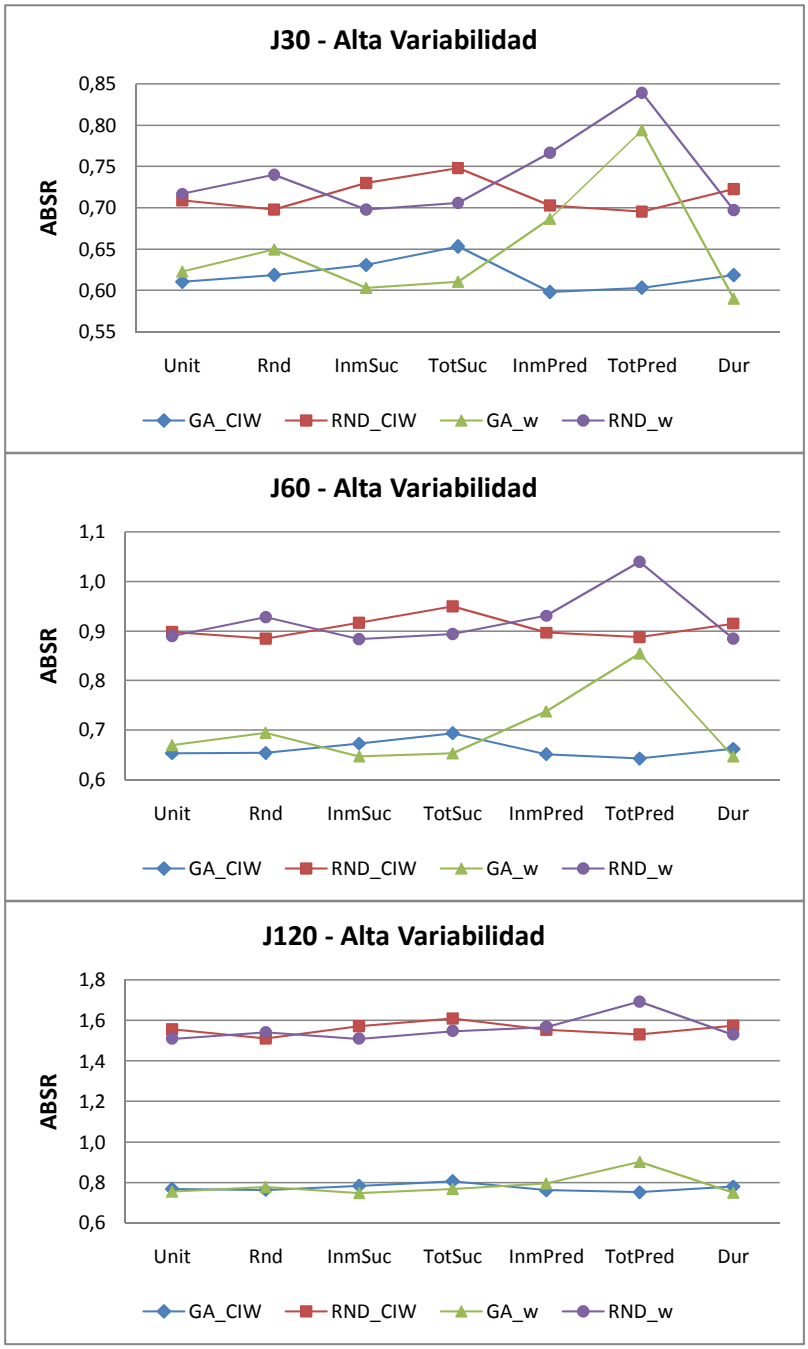


Figura 5.5. ABSR, escenario de alta variabilidad

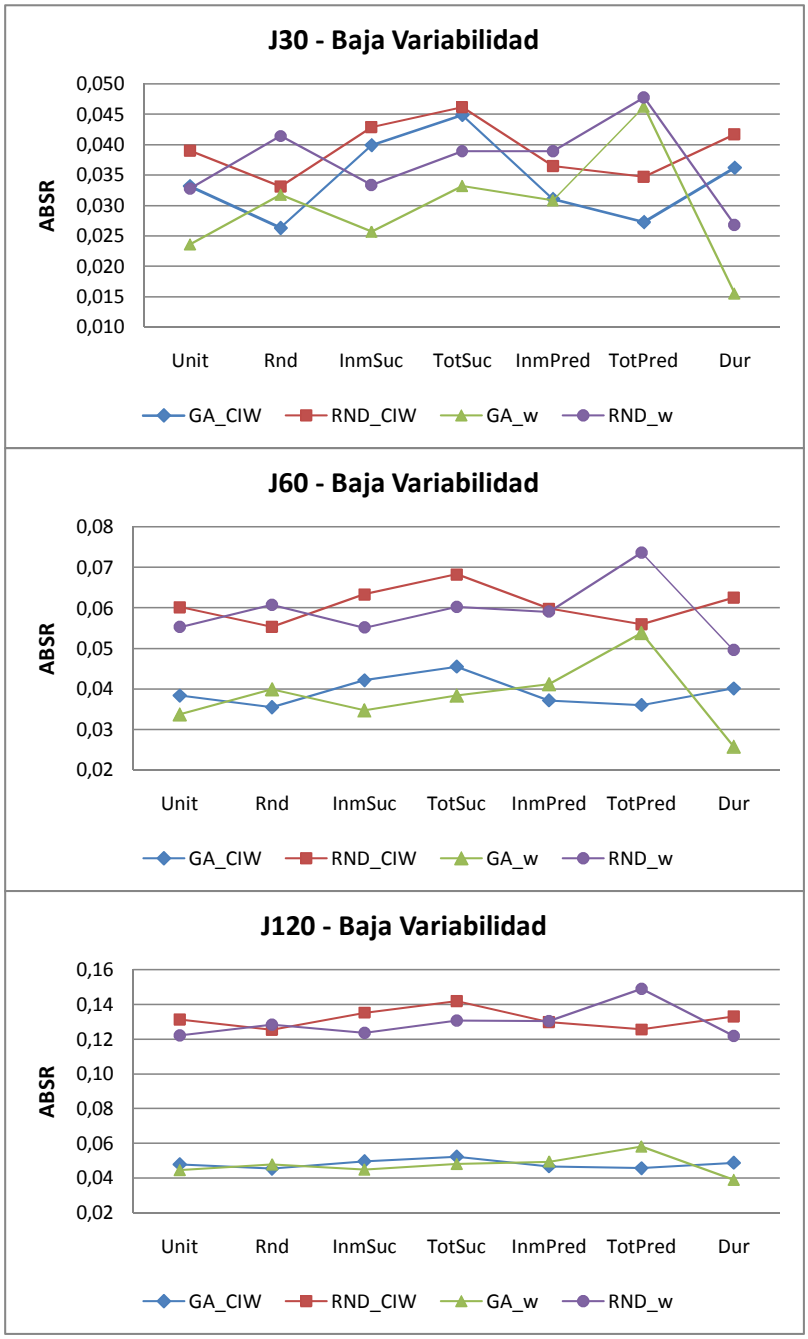


Figura 5.6. ABSR, escenario de baja variabilidad

Tabla 5.3. Resultados de la métrica ABSR en los dos escenarios de variabilidad

	Alta Variabilidad				Baja Variabilidad			
	GA_CIW	GA_w	RND_CIW	RND_w	GA_CIW	GA_w	RND_CIW	RND_w
J30								
Unit	0.611	0.623	0.709	0.717	0.033	0.024	0.039	0.033
Rnd	0.619	0.650	0.698	0.740	0.026	0.032	0.033	0.041
InmSuc	0.631	0.603	0.730	0.698	0.040	0.026	0.043	0.033
TotSuc	0.653	0.611	0.749	0.707	0.045	0.033	0.046	0.039
InmPred	0.598	0.687	0.703	0.767	0.031	0.031	0.037	0.039
TotPred	0.604	0.794	0.696	0.840	0.027	0.046	0.035	0.048
Dur	0.619	0.590	0.723	0.698	0.036	0.016	0.042	0.027
J60								
Unit	0.654	0.670	0.898	0.890	0.038	0.034	0.060	0.055
Rnd	0.654	0.695	0.885	0.928	0.036	0.040	0.055	0.061
InmSuc	0.673	0.647	0.917	0.884	0.042	0.035	0.063	0.055
TotSuc	0.694	0.653	0.950	0.894	0.046	0.038	0.068	0.060
InmPred	0.652	0.737	0.897	0.931	0.037	0.041	0.060	0.059
TotPred	0.643	0.855	0.888	1.040	0.036	0.054	0.056	0.074
Dur	0.662	0.647	0.915	0.885	0.040	0.026	0.063	0.050
J120								
Unit	0.768	0.756	1.559	1.510	0.048	0.045	0.131	0.122
Rnd	0.762	0.779	1.512	1.542	0.045	0.048	0.125	0.128
InmSuc	0.784	0.748	1.572	1.510	0.050	0.045	0.135	0.124
TotSuc	0.807	0.769	1.611	1.549	0.052	0.048	0.142	0.131
InmPred	0.761	0.796	1.555	1.569	0.047	0.049	0.130	0.131
TotPred	0.751	0.901	1.533	1.694	0.046	0.058	0.126	0.149
Dur	0.780	0.750	1.575	1.530	0.049	0.039	0.133	0.122

función de pesos acumulados modificada (CIW) para mejorar el desempeño de la secuencia base para finalizar antes del plazo estipulado y por tanto mejorando el %ADPC (Fig 5.7).

De otra parte, en el escenario de baja variabilidad (Figura 5.8) los pesos derivados del Número de Sucesores Totales son los que obtienen peores resultados, sin importar el método utilizado o si el peso es w o CIW.

Finalmente, la influencia de los pesos en la finalización del proyecto se resume para los tres conjuntos de problemas y los dos escenarios de variabilidad en la tabla 5.4. Para esta medida de robustez, la función de aptitud que utiliza los pesos de las actividades obtiene mejores resultados que aquella que utiliza el peso acumulado modificado sin importar si el método utilizado es el GA o el procedimiento aleatorio.

5.3.7. Coste de Estabilidad

Para medir el coste de estabilidad de la programación generada se utiliza el peso de cada actividad y que es una medida del coste por unidad de tiempo

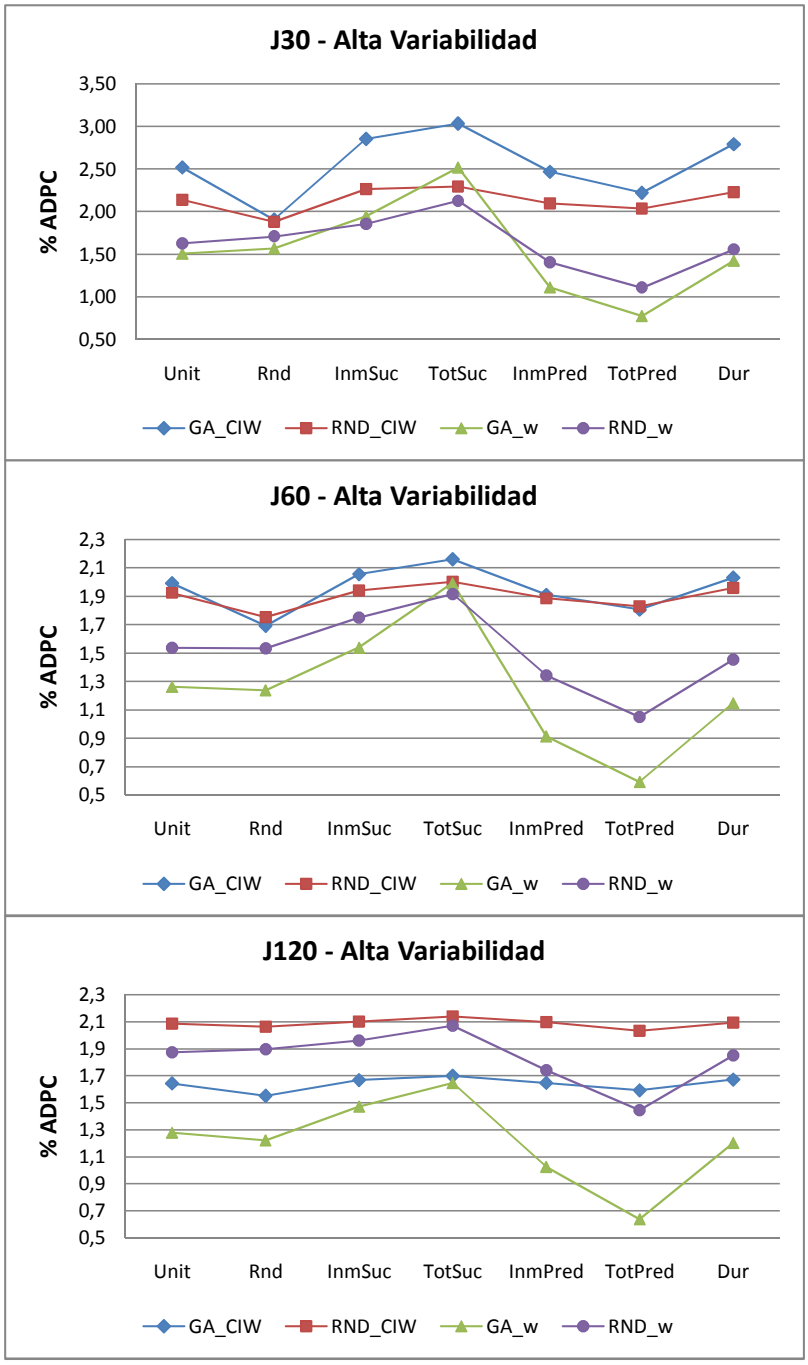


Figura 5.7. %ADPC, escenario de alta variabilidad

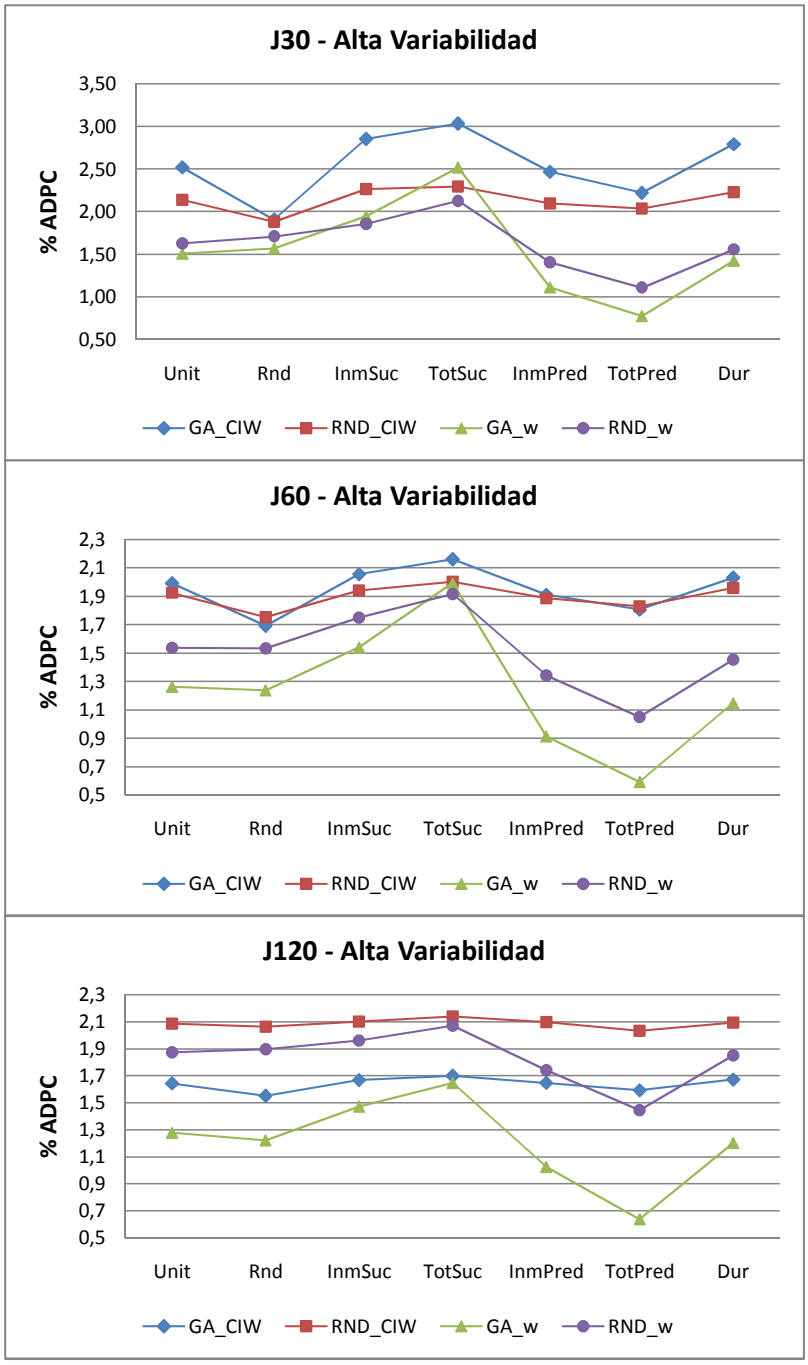


Figura 5.8. %ADPC, escenario de baja variabilidad

Tabla 5.4. Resultados de la medida %ADPC con alta y baja variabilidad

	Alta Variabilidad				Baja Variabilidad			
	GA_CIW	GA_w	RND_CIW	RND_w	GA_CIW	GA_w	RND_CIW	RND_w
J30								
Unit	2.518	1.505	2.138	1.278	0.185	0.044	0.133	0.059
Rnd	1.905	1.566	1.883	1.223	0.086	0.072	0.092	0.084
InmSuc	2.853	1.946	2.265	1.472	0.253	0.093	0.152	0.095
TotSuc	3.034	2.518	2.295	1.647	0.264	0.185	0.149	0.133
InmPred	2.468	1.112	2.101	1.026	0.186	0.021	0.125	0.041
TotPred	2.221	0.775	2.035	0.638	0.151	0.008	0.123	0.018
Dur	2.791	1.426	2.231	1.204	0.249	0.035	0.147	0.048
J60								
Unit	1.995	1.264	1.926	1.539	0.136	0.050	0.131	0.082
Rnd	1.695	1.239	1.754	1.534	0.097	0.051	0.109	0.075
InmSuc	2.058	1.540	1.942	1.753	0.142	0.083	0.128	0.105
TotSuc	2.164	1.995	2.004	1.918	0.146	0.136	0.135	0.131
InmPred	1.914	0.913	1.886	1.346	0.124	0.026	0.124	0.058
TotPred	1.809	0.593	1.829	1.052	0.119	0.008	0.118	0.024
Dur	2.034	1.150	1.959	1.456	0.143	0.038	0.136	0.066
J120								
Unit	1.644	1.278	2.087	1.874	0.090	0.054	0.125	0.101
Rnd	1.553	1.223	2.064	1.898	0.082	0.049	0.127	0.098
InmSuc	1.668	1.472	2.103	1.963	0.092	0.074	0.131	0.115
TotSuc	1.699	1.647	2.139	2.072	0.088	0.090	0.129	0.125
InmPred	1.648	1.026	2.096	1.742	0.091	0.037	0.128	0.083
TotPred	1.594	0.638	2.033	1.445	0.091	0.010	0.122	0.039
Dur	1.672	1.204	2.092	1.853	0.096	0.047	0.130	0.100

de comenzarla antes o después de su instante planeado. Van de Vonder et al. [101] y Lambrechts et al. [61] utilizan la ecuación 2.12 para calcular dicho coste.

Dado que esta función depende de los pesos asociados a cada actividad, no es posible comparar entre sí todos los resultados obtenidos entre los métodos desarrollados en este capítulo o con los trabajos desarrollados previamente por otros autores. Sin embargo, es posible comparar el coste de cada conjunto de pesos en los cuatro procedimientos desarrollados.

La tabla 5.5 muestra los costes de estabilidad para los tres conjuntos de la librería de pruebas. En todos los casos,los costes obtenidos por el GA son menores que los del procedimiento aleatorio. En el escenario de baja variabilidad, el procedimiento GA_w tiene un mejor desempeño que los otros procedimientos, obteniendo costes inferiores de manera consistente.

En el escenario de alta variabilidad el procedimiento GA_CIW obtiene en promedio un coste menor que el GA_w en el conjunto J30. En los conjuntos J60 y J120 sucede lo contrario.

Tabla 5.5. Coste de Estabilidad

	Alta Variabilidad				**Baja Variabilidad**			
J30								
	GA_CIW	RND_CIW	GA_w	RND_w	GA_CIW	RND_CIW	GA_w	RND_w
Unit	18.93	21.98	19.31	22.23	1.03	1.21	0.73	1.02
Rnd	120.98	130.70	117.56	132.90	5.21	6.31	5.73	7.18
InmSuc	25.96	33.05	27.96	33.32	1.34	1.78	1.12	1.56
TotSuc	76.62	107.71	81.36	106.68	3.53	5.76	3.07	5.12
InmPred	42.48	47.88	43.34	49.65	2.40	2.55	1.66	2.25
TotPred	235.02	247.78	211.40	246.18	13.04	13.37	7.17	9.85
Dur	91.87	111.23	94.81	111.28	4.83	5.99	2.49	4.23
J60								
	GA_CIW	RND_CIW	GA_w	RND_w	GA_CIW	RND_CIW	GA_w	RND_w
Unit	39.86	54.77	40.84	54.31	2.34	3.67	2.06	3.37
Rnd	201.38	255.39	196.74	259.39	11.05	15.90	10.69	16.24
InmSuc	60.74	87.83	63.68	87.54	3.43	5.84	3.32	5.50
TotSuc	284.95	451.41	292.90	444.16	13.93	29.81	12.81	27.92
InmPred	87.80	115.48	87.64	114.04	5.33	7.77	4.32	6.66
TotPred	736.34	904.77	637.77	894.73	47.09	58.01	25.75	47.78
Dur	206.79	289.40	209.11	283.55	12.19	19.43	8.31	15.68
J120								
	GA_CIW	RND_CIW	GA_w	RND_w	GA_CIW	RND_CIW	GA_w	RND_w
Unit	92.96	188.60	91.47	182.74	5.80	15.88	5.40	14.79
Rnd	397.63	741.76	389.53	747.32	22.89	58.73	22.45	58.46
InmSuc	148.93	309.12	149.02	301.27	9.22	26.90	8.99	25.29
TotSuc	1130.28	2417.88	1125.53	2356.99	67.97	219.04	64.62	207.58
InmPred	196.77	390.67	186.04	381.74	12.11	31.29	10.50	29.20
TotPred	2537.98	4631.22	2120.39	4613.21	152.69	337.14	95.42	309.07
Dur	487.78	997.71	472.63	971.00	30.27	83.40	24.22	76.32

5.3.8. Impacto de las características del proyecto

Finalmente, es interesante estudiar el impacto de las características del proyecto en el desempeño del algoritmo. Tal como se presentó en la sección 2.6.1 el generador ProGen utiliza tres características para generar las instancias: la complejidad de la red (NC), el Factor de Recursos (RF) y el grado de restricción de los recursos (RS). Para este estudio se utilizan los resultados del procedimiento AG_w.

Complejidad de la Red

La primera característica que examinamos es la complejidad de la red, la cual es definida como el número de arcos no redundantes por nodo, incluyendo las actividades ficticias.

En el escenario de alta variabilidad (figura 5.9) observamos el impacto de la complejidad de la red. En el escenario de baja variabilidad (figura 5.10) el impacto es especialmente evidente en el conjunto de 120 actividades. En todos casos, los problemas con valores inferiores de la complejidad de la red permiten la generación de secuencias estables. Este resultado es el esperado, al tener en cuenta que entre menos relacionadas estén las actividades más flexible pueden ser las programaciones, tal como se plantea en la medida de flexibilidad de Aloulu & Portman (2003) [5] presentada en la sección 2.4.1.

Como se observa en las figuras 5.11 y 5.12, el impacto de la complejidad de la red no es tan claro en la medida %ADPC como en el caso anterior. Sin embargo, de manera general mantiene que entre menor sea la complejidad de la red mayor es el porcentaje de proyectos terminados a tiempo. Lo que resulta determinante en este caso es el conjunto utilizado para los pesos de las actividades. Al utilizar como factor de ponderación los predecesores totales de la actividad el porcentaje de proyectos finalizados a tiempo es similar para todos los valores de la complejidad de la red.

Factor de Recurso

La segunda característica analizada es el Factor de Recurso (RF) el cual refleja la proporción promedio de recursos utilizados por una actividad. Un valor de cero indica que la actividad no requiere ningún recurso. Por el contrario, si RF=1, entonces cada actividad require usar todos los tipos de recursos.

Como cabe esperar, la relación entre el Factor de Recurso y las dos medidas de robustez propuesta no es evidente. Las figuras 5.13 y 5.14 muestra el impacto sobre la robustez de la solución. Para el conjunto J30 los problemas más fáciles de resolver son aquellos donde RF=0.25, mientras que para J120 es RF=1, lo cual no deja de ser sorprendente. Sin embargo, esto se explica porque el Factor de Recurso no hace referencia a la relación a la cantidad de los recursos demandados y su disponibilidad. En los tres conjuntos los problemas con RF=0.5 presentan una mayor dificultad para obtener secuencias base robustas.

Las figuras 5.15 y 5.16 muestran el impacto de RF en la finalización del proyecto. Es evidente que el factor RF no es un buen discriminador de la dificultad de los problemas, al menos con el método utilizado. Sólo en el caso del conjunto J120 y en el escenario de alta variabilidad hay una distinción más clara en el impacto.

Grado de Restricción de los Recursos

Finalmente se analiza el grado de restricción de los recursos (RS) el cual expresa la relación entre la demanda de los recursos y su disponibilidad. Cuando RS=1 el problema no tiene restricción de recursos.

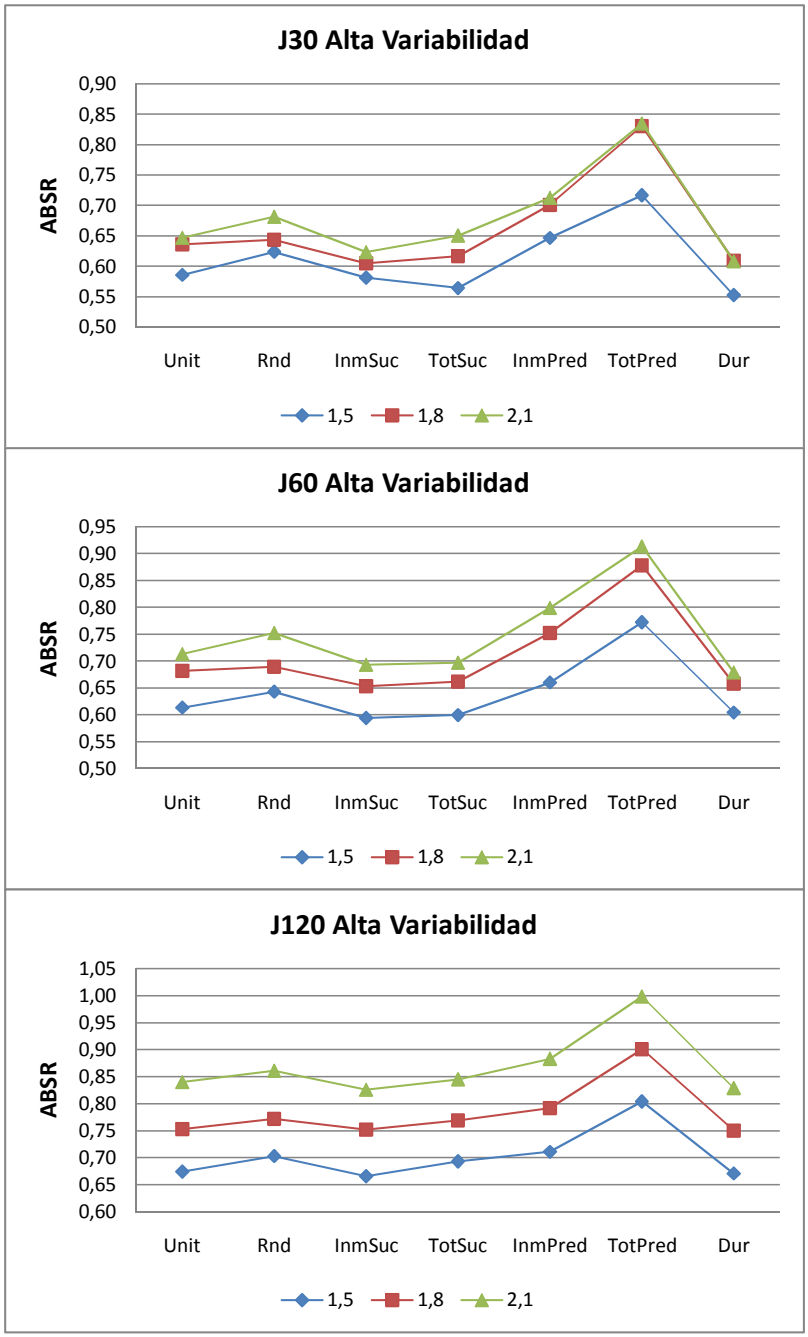


Figura 5.9. Impacto de la Complejidad de la Red en la medida ABSR - escenario de alta variabilidad

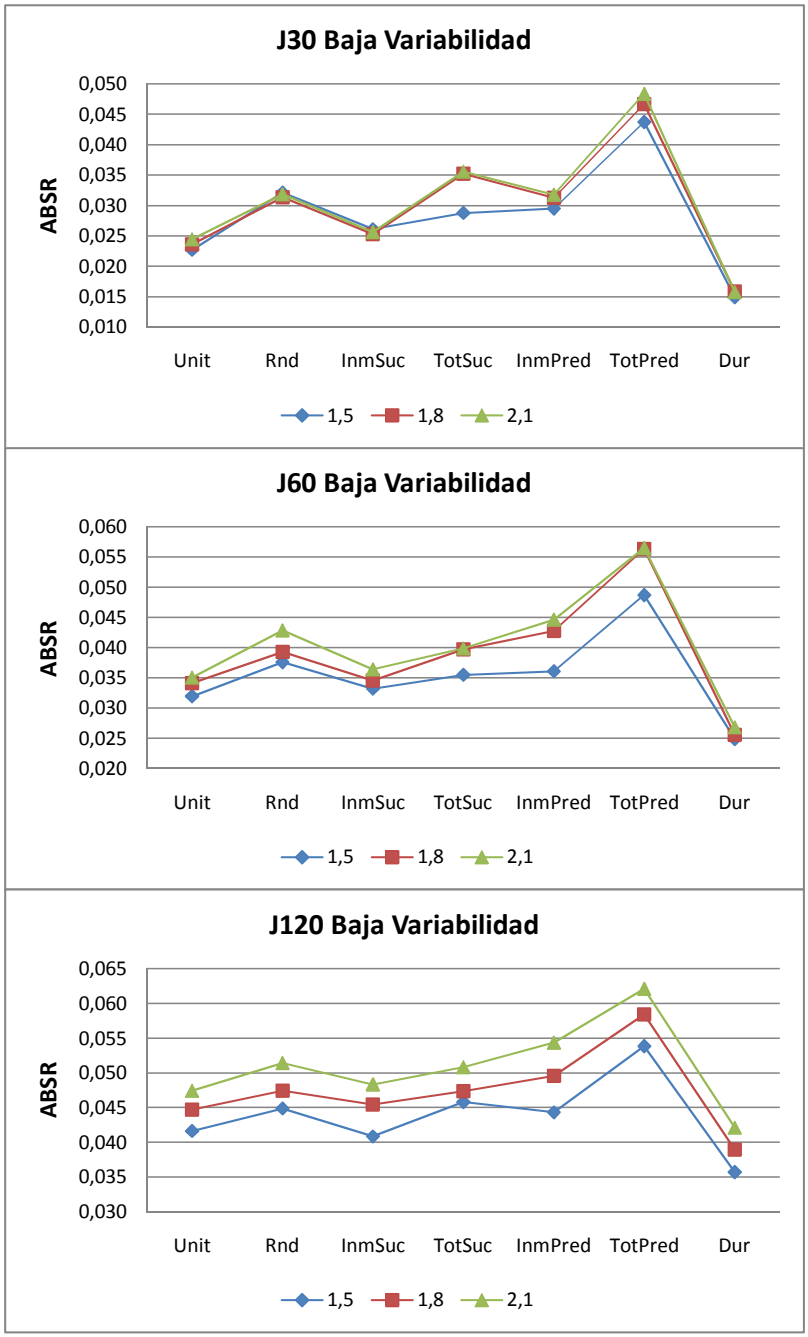


Figura 5.10. Impacto de la Complejidad de la Red en la medida ABSR - escenario de baja variabilidad

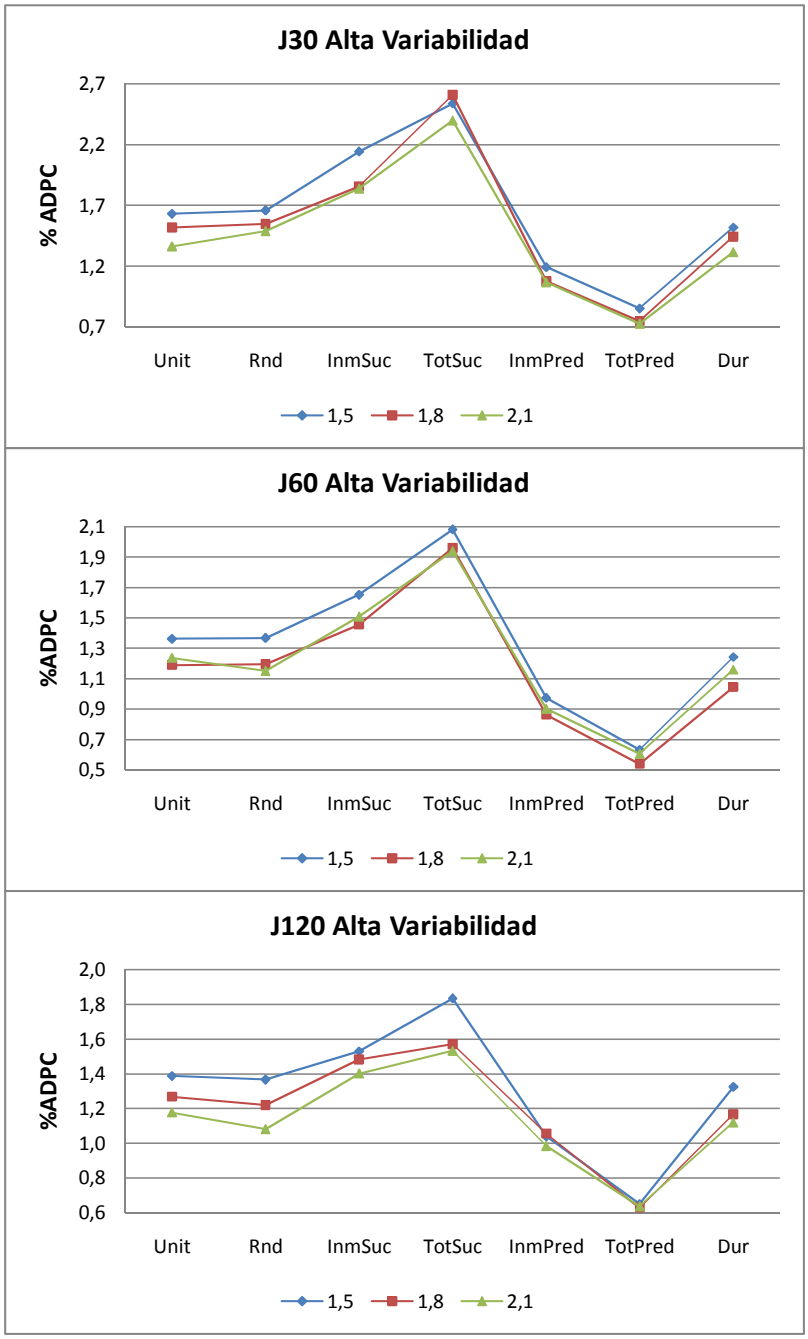


Figura 5.11. Impacto de la complejidad de la red en la medida %ADPC - escenario de alta variabilidad

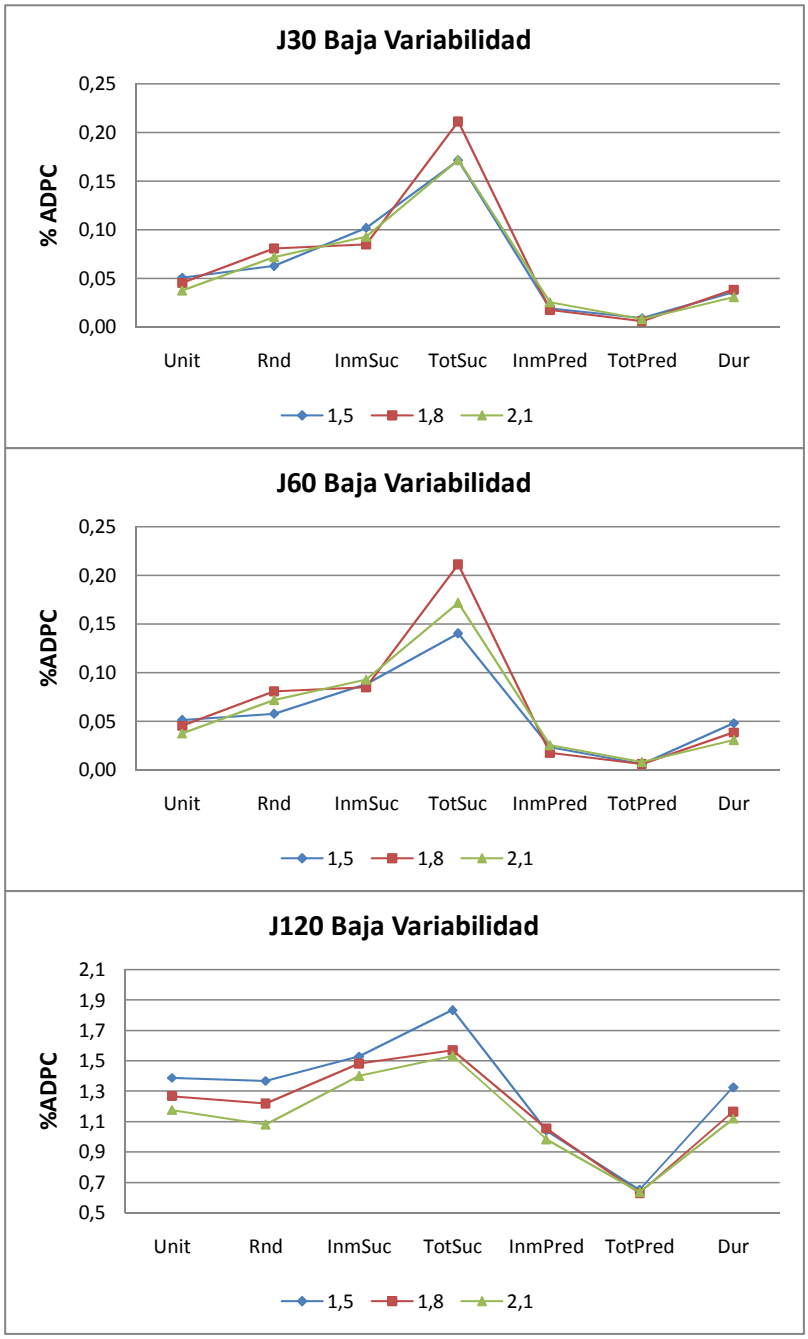


Figura 5.12. Impacto de la complejidad de la red en la medida %ADPC - escenario de baja variabilidad

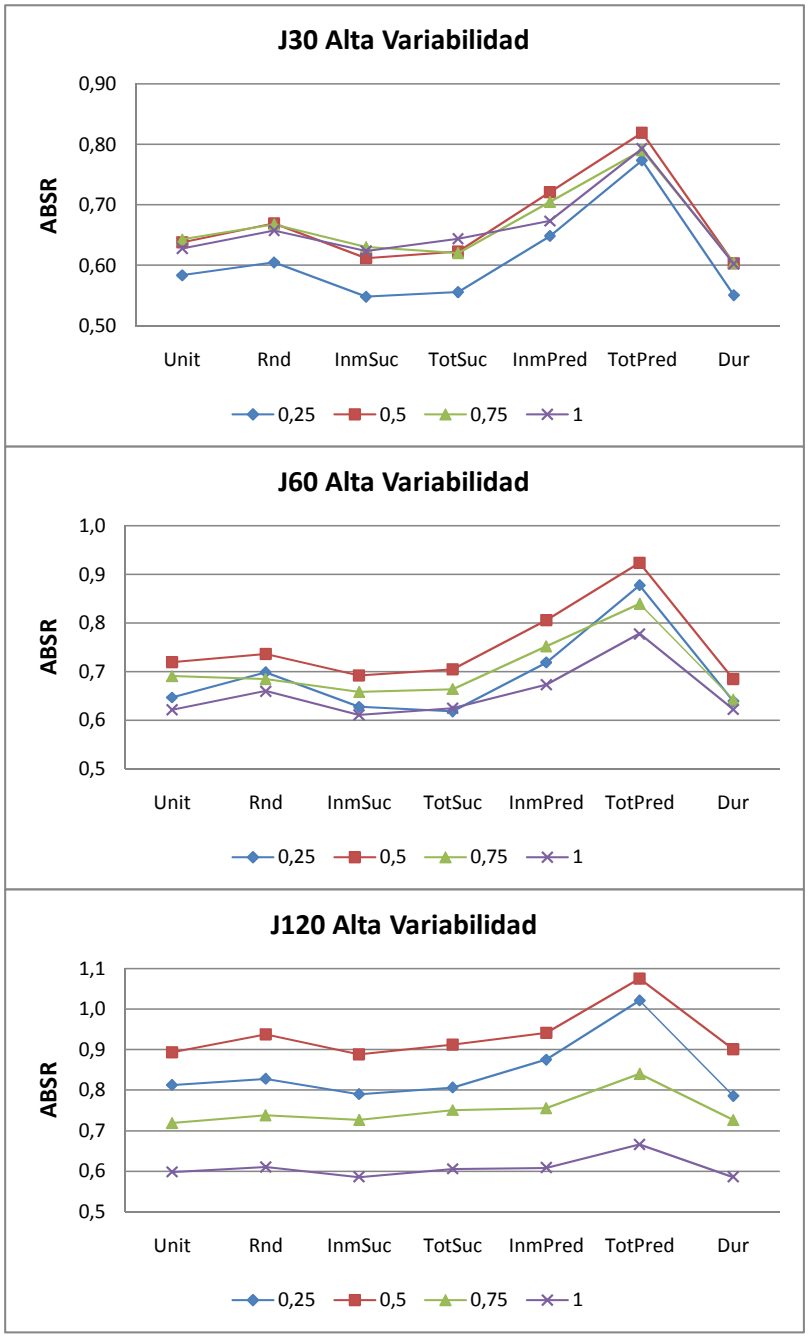


Figura 5.13. Impacto del Factor de Recurso en la medida ABSR - escenario de alta variabilidad

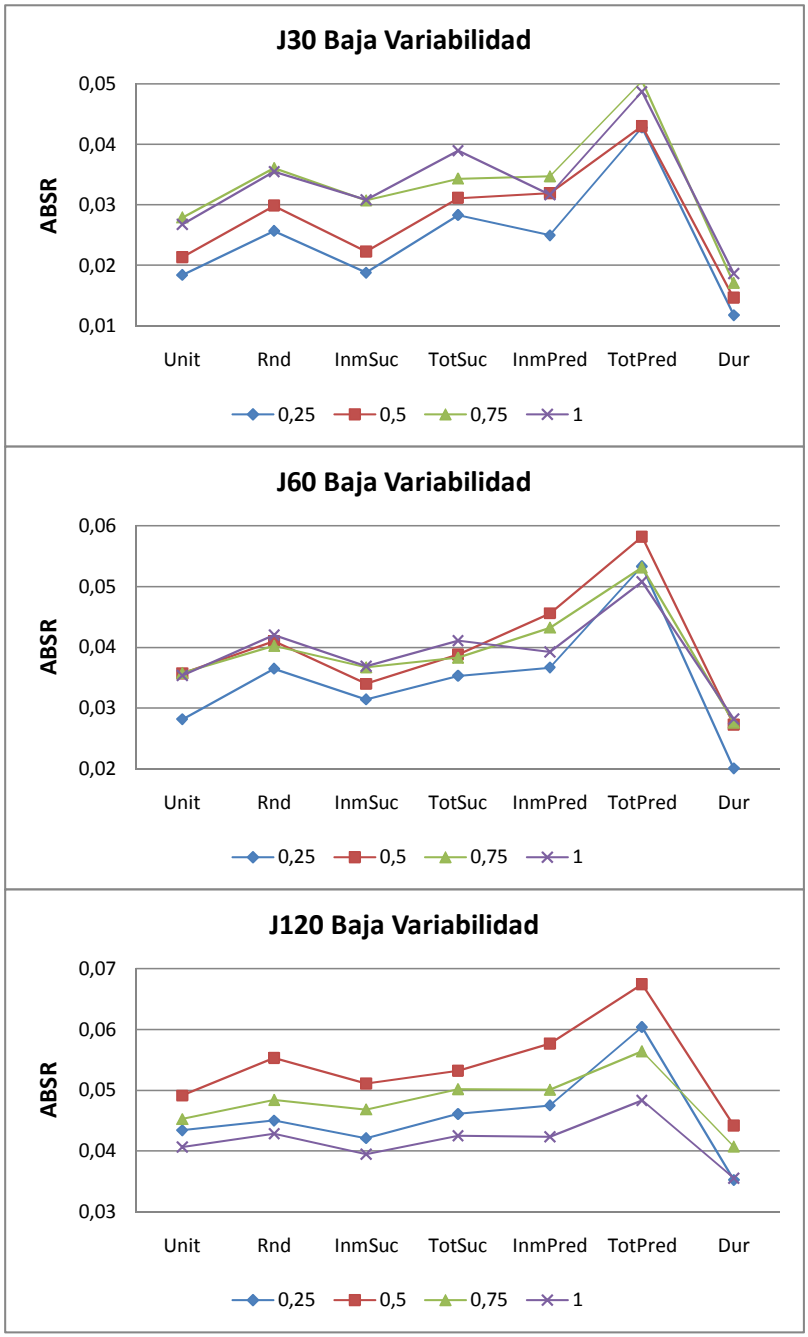


Figura 5.14. Impacto del Factor de Recurso en la medida ABSR - escenario de baja variabilidad

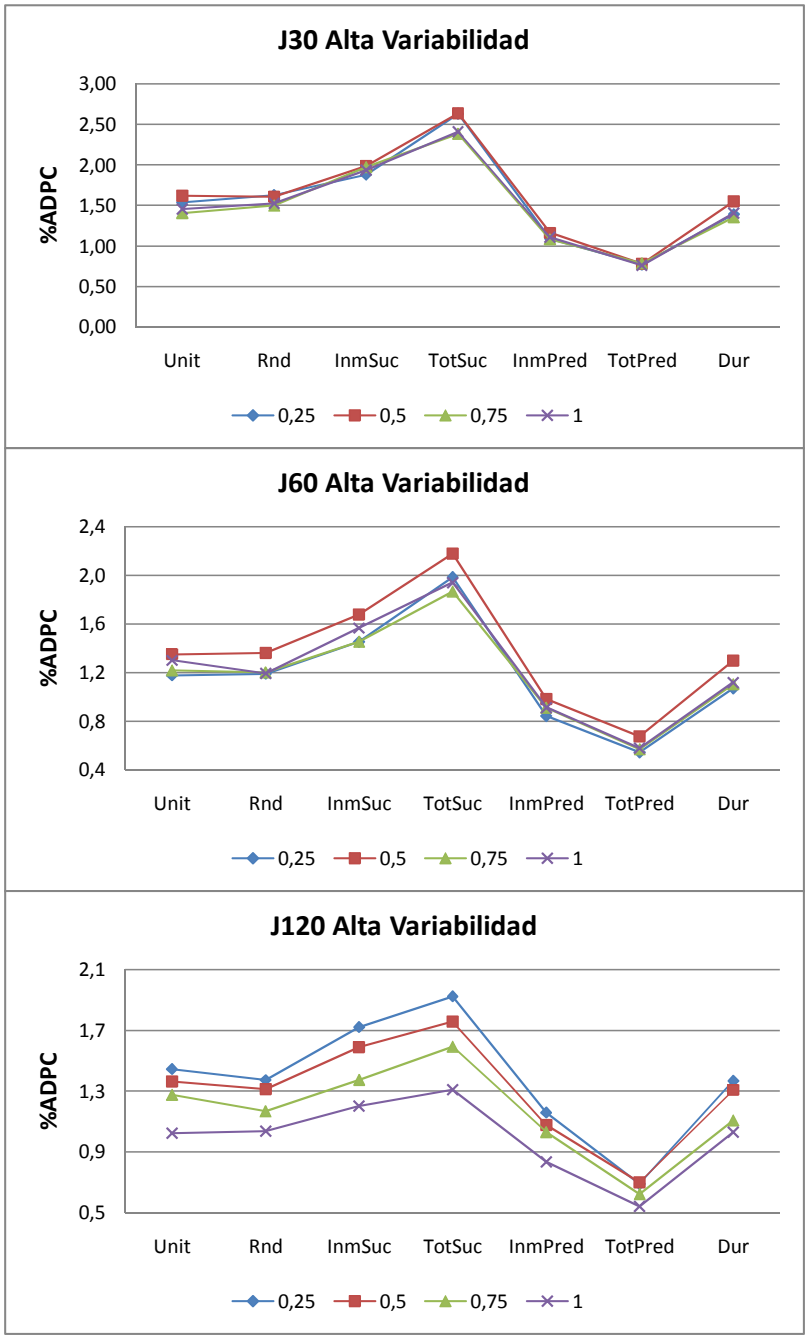


Figura 5.15. Impacto del Factor de Recurso en la medida %ADPC - escenario de alta variabilidad

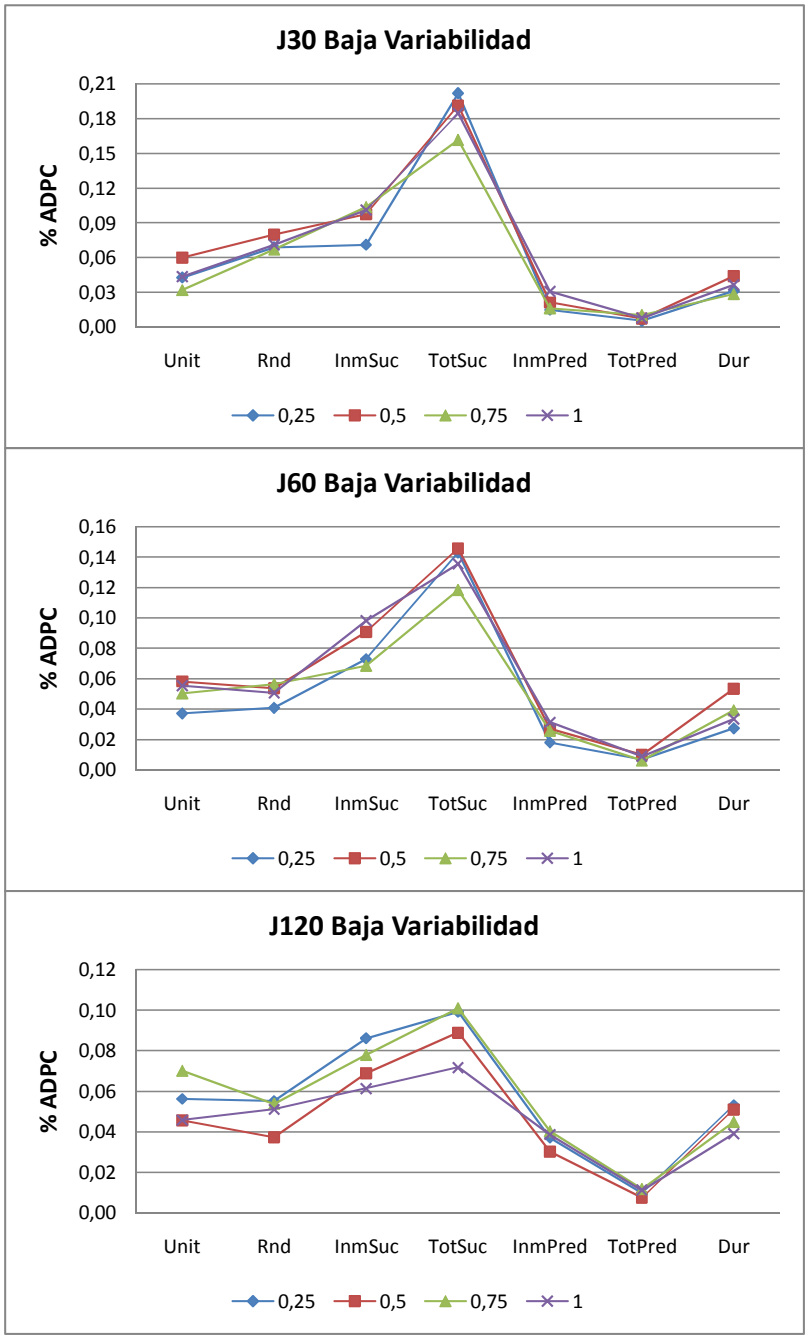


Figura 5.16. Impacto del Factor de Recurso en la medida %ADPC - escenario de baja variabilidad

Como cabe esperar, cuando se evalúa la robustez de la solución, los problemas con mayor RS no logran recuperarse de las disrupciones tan fácilmente como aquellos que son menos restringidos. Los valores de inestabilidad son mayores en el escenario de alta variabilidad (figura 5.17) que en el de baja variabilidad (figura 5.18), pero los comportamientos son similares. Sin embargo, la estrategia utilizada para seleccionar los pesos es también determinante a la hora de generar secuencias base robustas.

En concreto, los pesos derivados de la duración de las actividades tienen el mejor desempeño. Esto es consecuencia de la relación entre la variabilidad de las actividades en el simulador y la duración media que se utiliza como valor determinístico de la planeación. Es decir, en el escenario de alta variabilidad una actividad con duración media $d_j = 5$ varía entre 1.25 y 14.375, mientras que una actividad con duración $d_j = 10$ varía entre 2.5 y 28.75.

Si bien es cierto que al utilizar la distribución beta con parámetros 2 y 5 es más probable obtener duraciones cercanas a la media o menores a ella, como empleamos la política de *railway scheduling* en el método reactivo estos adelantos no se reflejan en la programación. Se escoge esta política, porque el coste de inestabilidad penaliza empezar tarde o temprano la actividad.

Entonces, una actividad con una mayor duración tiene mayores probabilidades de sufrir mayores disrupciones y desestabilizar en mayor grado la programación planeada.

Las figuras 5.19 y 5.20 muestran el impacto de los diferente conjuntos de pesos en la finalización del proyecto. Los pesos tomados del Número de Predecesores totales son los que obtienen mejores resultados sin importar el grado de restricción de los recursos.

5.4. Conclusiones

En este capítulo hemos abordado el problema de generación de programaciones robustas. Como método de generación de programaciones robustas proponemos un algoritmo genético. En una primera fase se resuelve el RCPSP tomando la duración media de las actividades y luego inserta tiempos de seguridad para evitar la propagación de las disrupciones a lo largo de la programación. Asumimos que si una programación es robusta la variación entre el tiempo planeado y el tiempo realizado será cercana a cero.

Dada la complejidad de la evaluación *a priori* de la robustez de una programación, la función de fitness de los individuos se hace mediante una medida surrogada de la robustez. Esta medida se basa en el aporte de los buffers en la estabilidad de la programación y se pondera con el impacto de la actividad en la robustez de la secuencia. Se estudia la relación de las características de la actividad con su contribución a la robustez de la solución.

Las programaciones se someten a un proceso que simula su ejecución en un entorno de alta y baja variabilidad. Para medir la estabilidad de las programaciones proponemos dos medidas de robustez. Una enfocada en la

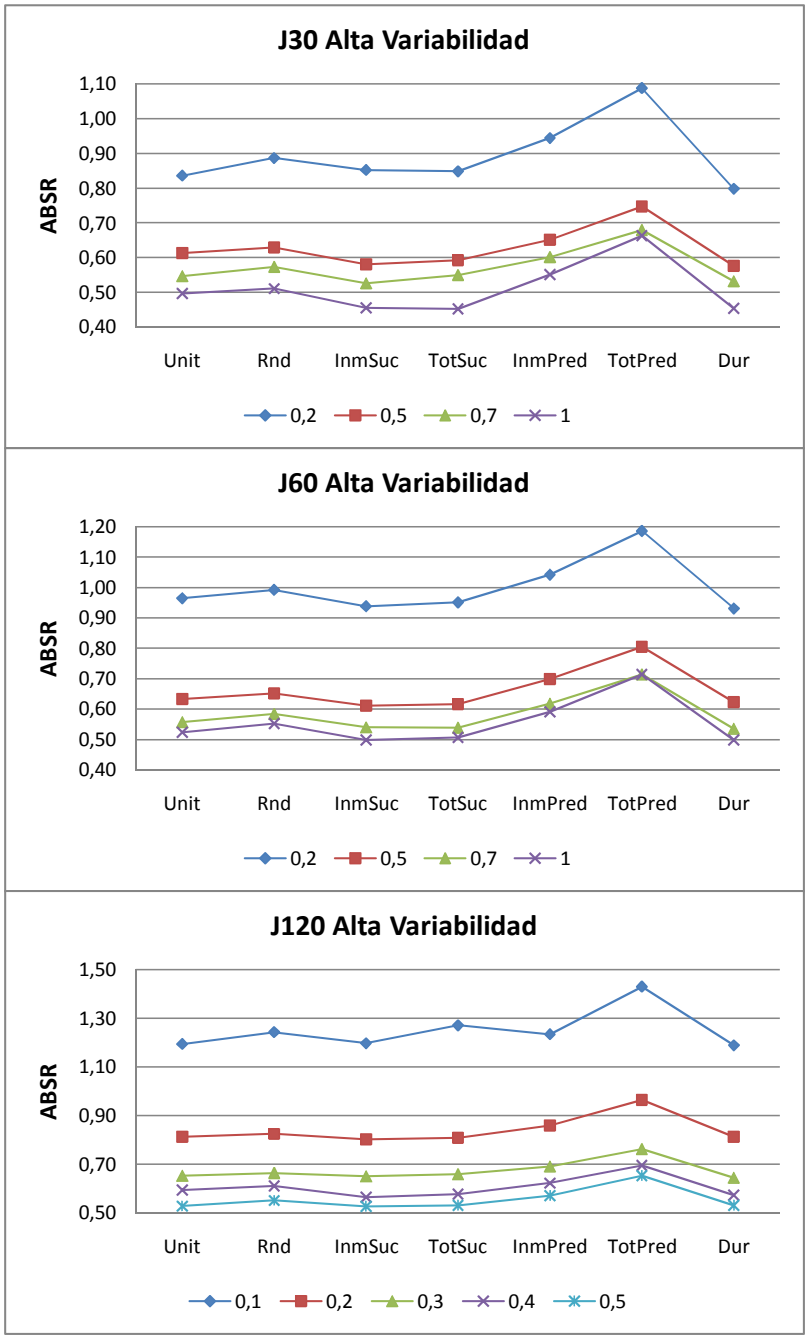


Figura 5.17. Impacto del Grado de Restricción de Recursos en la medida ABSR - escenario de alta variabilidad

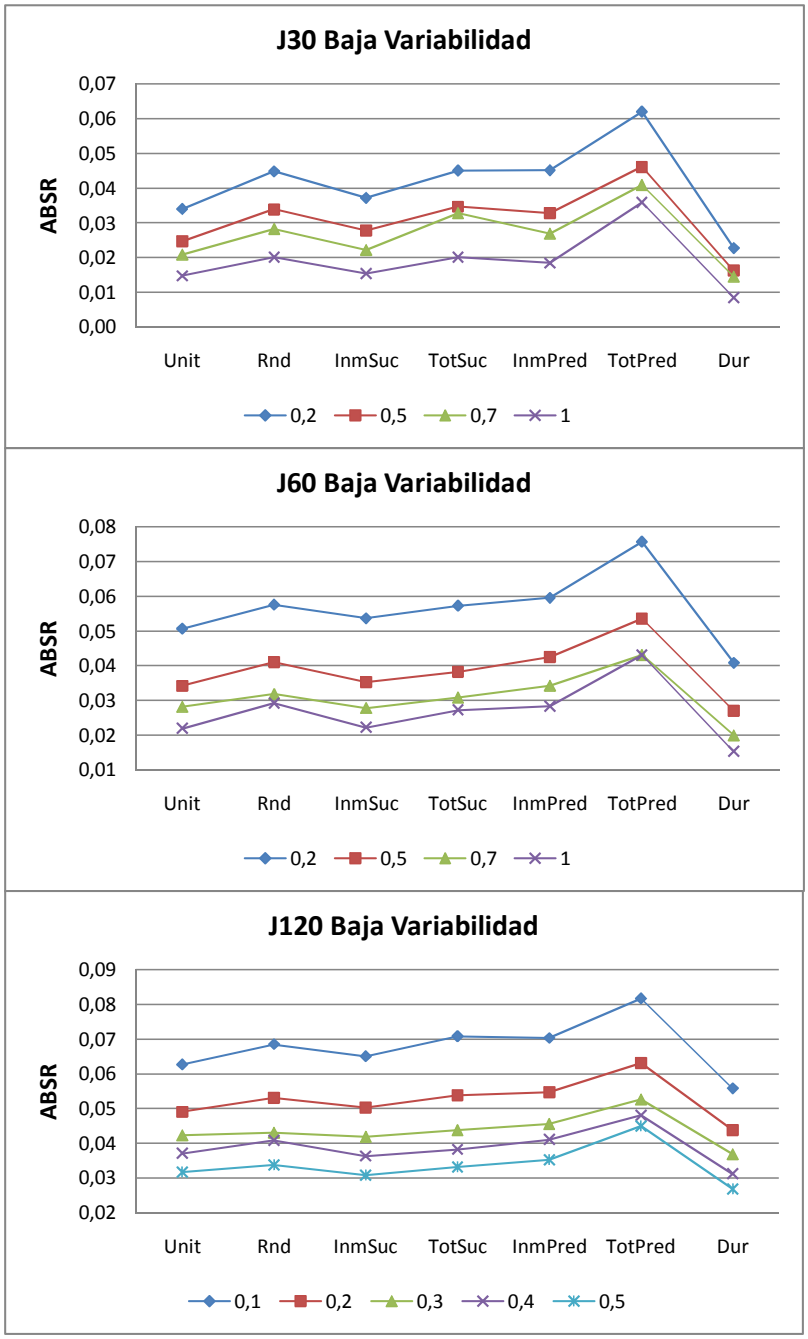


Figura 5.18. Impacto del Grado de Restricción de Recursos en la medida ABSR - escenario de baja variabilidad

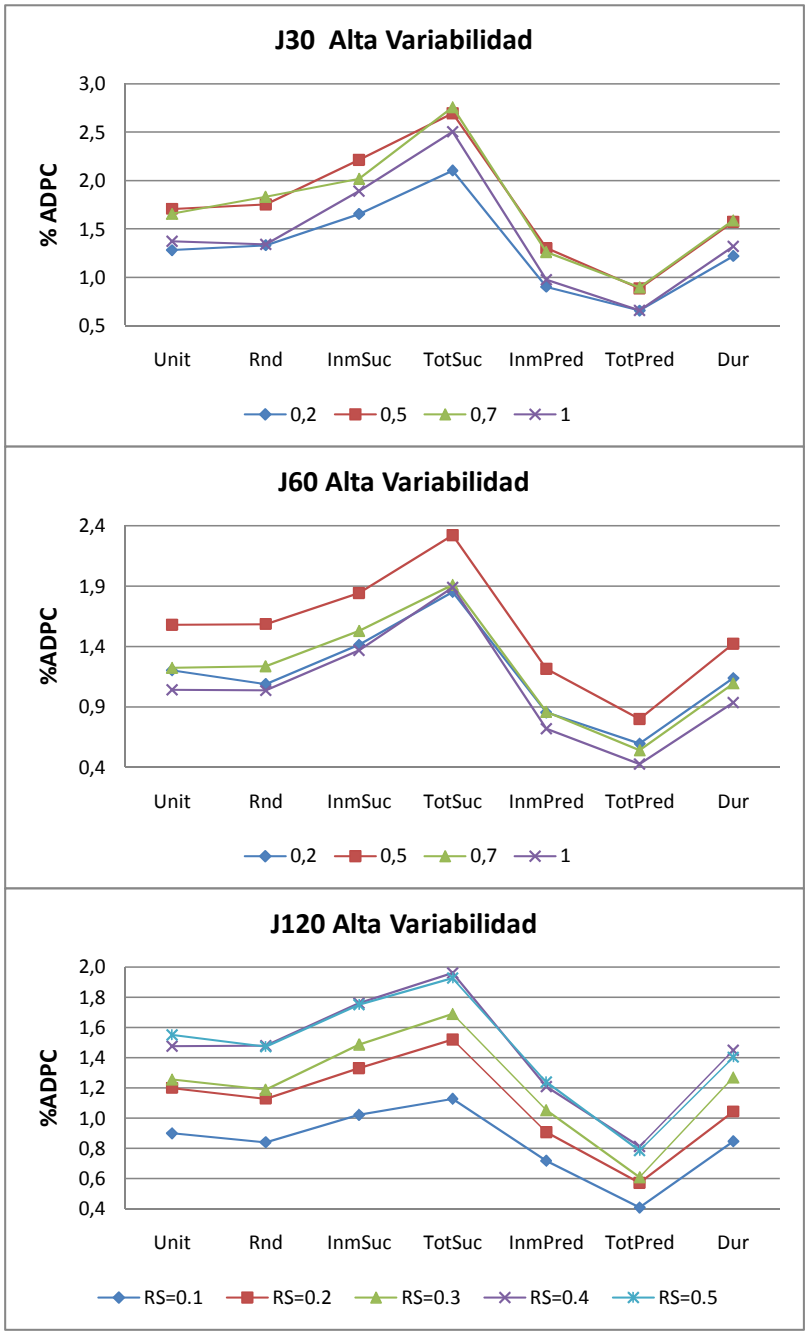


Figura 5.19. Impacto del Grado de Restricción de Recursos en la medida ABSR - escenario de alta variabilidad

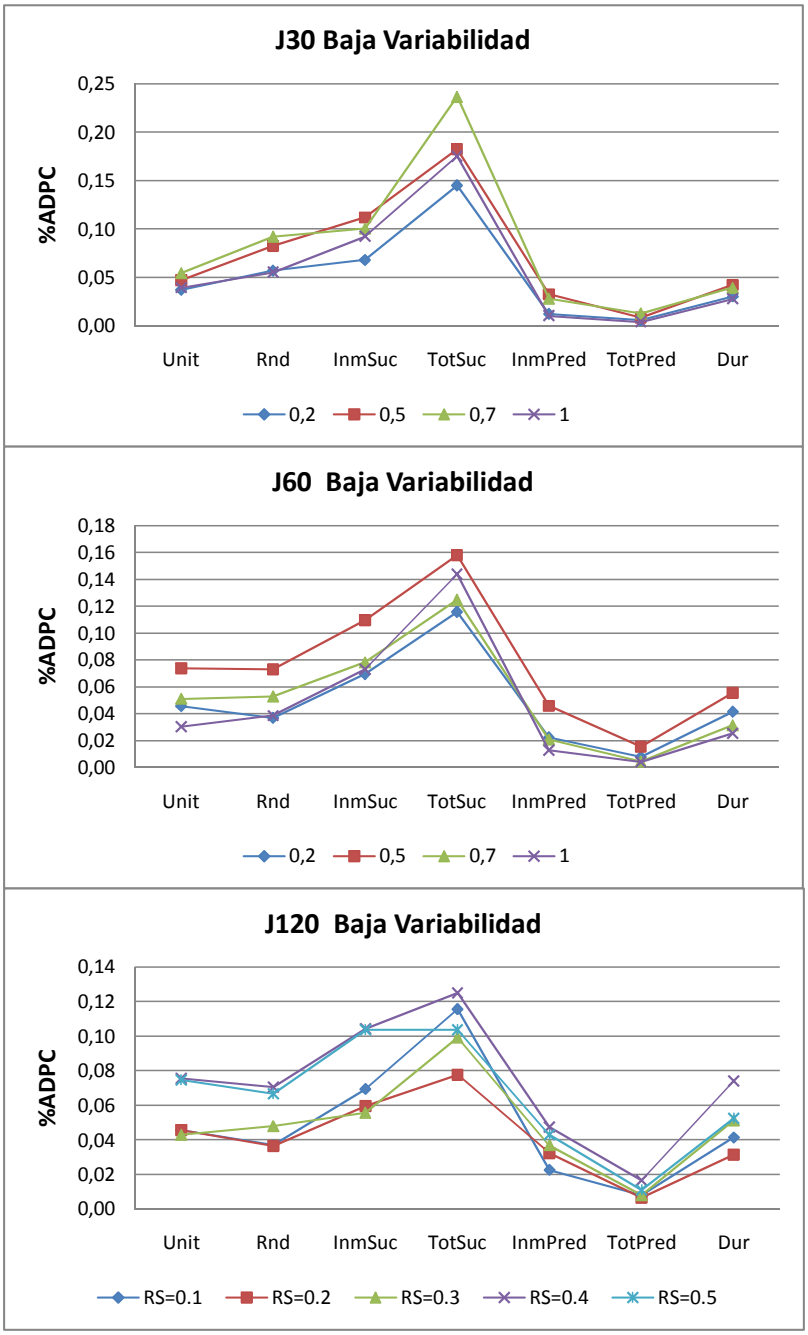


Figura 5.20. Impacto del Grado de Restricción de Recursos en la medida ABSR - escenario de baja variabilidad

estabilidad de la secuencia y la otra en la finalización del proyecto. En cuanto al porcentaje de retraso total del proyecto, la característica de la actividad que está más relacionada con la importancia de la actividad en la robustez de la programación es la duración. En cuanto al retraso promedio de las actividades, el Número Total de Predecesores es la que obtiene mejores resultados.

En cuanto al análisis de las características de los proyectos, la complejidad de la red (NC) y el grado de restricción de los recursos (RS) afectan directamente la estabilidad de la secuencia. En cuanto más alta es la complejidad de la red es más difícil que la programación se recupere de una disrupción, esto se debe a que las actividades están más relacionadas entre sí disminuyendo la flexibilidad de la programación. Lo mismo sucede cuando los recursos son altamente limitados. Las características de los proyectos no tienen un impacto tan claro en cuanto a la finalización del proyecto.

6

Conclusiones

En esta tesis hemos abordado el problema de la Programación de Proyectos con Recursos Limitados, tanto en su versión estándar o modo único como en el caso multi-modo. Así mismo analizado el caso de la generación de programaciones robustas. Estos problemas han sido estudiados en la literatura y se han desarrollado diferentes tipos de técnicas para resolverlos.

A continuación se resumen las aportaciones y conclusiones realizadas en esta tesis con relación a los objetivos planteados inicialmente. Así mismo se presentan las líneas futuras de investigación a seguir y las publicaciones que han sido fruto del trabajo de investigación realizado.

6.1. Aportaciones

En primer lugar hemos realizado una revisión bibliográfica de los distintos métodos publicados para resolver cada uno de los problemas. Para el problema RCPSP y el problema MRCPSP existen diferentes algoritmos exactos que lo resuelven pero que no son aplicables a muchos problemas. La alternativa a estos métodos es la constituyen los algoritmos heurísticos, de los cuales los basados en las reglas de prioridad fueron los primeros en ser utilizados. En la actualidad los métodos metaheurísticos son los que mejores resultados obtienen.

El enfoque que hemos escogido está basado en los algoritmos genéticos debido a que los resultados que se han reportado en la literatura los muestran como promisorios para hallar soluciones de alta calidad aún en problemas de grandes dimensiones. De acuerdo con los objetivos planteados, la principal aportación de esta tesis doctoral es la propuesta de métodos de asignación optimizada de recursos para el problema de programación de proyectos en cada una de las versiones estudiadas. A continuación detallamos las aportaciones en cada uno de los problemas estudiados.

6.1.1. RCPSP

Para el problema RCPSP hemos partido del estudio del impacto del método de mejora FBI en los resultados obtenidos por las diversas reglas de prioridad. Hemos concluido que este proceso de búsqueda produce mejores resultados cuando la programación base es de mejor calidad. Las reglas de prioridad de Mínimo Tiempo Tardío de Finalización (MinLFT), Mínimo Tiempo Tardío de Iniciación (MinLST) y Máximo Número de Sucesores Totales (MaxSucTot) generan las mejores soluciones y las peores soluciones son generadas por las reglas de Mínima Duración y Uso Mínimo de Recursos.

Como resultado de estudio sobre el grado de aleatorización concluimos que en problemas pequeños vale la pena hacer un muestreo del espacio de soluciones y en problemas grandes es mejor buscar cerca de donde están las buenas soluciones.

El método heurístico propuesto para el RCPSP utiliza de manera selectiva el método de mejora FBI de tal manera que el esfuerzo invertido en él resulte provechoso para la calidad de la solución final.

Este procedimiento heurístico es competitivo con métodos metaheurísticos diseñados para este problema. En particular, es más eficiente para resolver los problemas de gran tamaño (J120) y es muy eficiente en cuanto a los tiempos de cómputo.

Los algoritmos genéticos propuestos incorporan el uso de las mejores reglas de prioridad y el método aleatorizado para explorar zonas promisorias del espacio de solución. La búsqueda local que se hace mediante el método FBI incluye un refinamiento adicional y es que no sólo se tiene en cuenta la calidad de la solución de partida sino que se tiene en cuenta el grado de profundidad de la búsqueda. Entre más soluciones se hayan generado más iteraciones FBI se realizarán.

Se ha realizado un estudio computacional para probar los efectos de incluir una población estática o dinámica y el tipo de sustitución de la población. El objetivo de incorporar una población dinámica es evitar la dependencia que existe en los métodos actuales de relacionar el tamaño de la población con el número de soluciones generadas. Lo cual es un supuesto poco realista en casos prácticos. Los algoritmos con sustitución inmediata (steady-state) demostraron obtener mejores resultados que los de sustitución poblacional. El algoritmo genético propuesto está entre los mejores desarrollados para la resolución del RCPSP.

6.1.2. MRCPSP

En el caso del MRCPSP la aportación más importante es la extensión del método de mejora FBI al caso de múltiples modos (MM-FBI). Este nuevo método, al igual que el FBI aprovecha la disponibilidad de recursos y la holgura libre de las actividades para compactar la solución. Adicionalmente, explota que cada actividad tiene múltiples modos de ejecución y valora

la bondad del cambio de modo en términos de la compactación de la programación parcial, siempre y cuando no se exceda la capacidad de los recursos disponibles.

Por otra parte, para el problema de la selección de modos diseñamos un método basado en el uso normalización de los recursos no renovables (MNR). Este método permite escoger el modo que utiliza menos recursos pero de manera proporcional a la disponibilidad de los mismos, lo cual hace que el porcentaje de asignaciones factibles sea bastante alto. De manera más precisa, éste proceso obtiene soluciones factibles en un rango que varía del 95.61 % al 98.73 %, mientras que el método que selecciona el modo con mayor duración (que generalmente utiliza menos recursos) no supera el 95.29 % de los casos. Finalmente, la selección aleatoria de modos es la que peor desempeño tiene y no llega nunca al 60 % de asignaciones factibles. La población inicial del Algoritmo Genético propuesto se basa en el uso aleatorizado de este método de asignación de modos.

Se debe tener en cuenta que para el correcto funcionamiento de un Algoritmo Genético es necesario el cálculo de una función objetivo o de aptitud que guíe la evolución de las generaciones a fin de obtener las soluciones esperadas. En el caso del MRCPSP no puede utilizarse únicamente la duración del proyecto, sino que debe tenerse en cuenta la satisfacción o violación de la capacidad de los recursos no renovables.

Para ello se genera una nueva función de evaluación que no se deja distorsionar por las diferentes magnitudes de la duración del proyecto y el exceso de los recursos no renovables. Con esta función, los individuos factibles siempre tienen un mejor desempeño que los no factibles, y éstos se diferencian entre sí por el exceso de recursos utilizados, lo cual guía de manera efectiva al algoritmo en la búsqueda de buenas soluciones. Esto se evidencia en los resultados obtenidos al compararla con otras funciones reportadas en la literatura. La nueva función de fitness logra mejores resultados con un nivel de confianza del 90 %

Un último aspecto a destacar es el nuevo operador de mutación diseñado específicamente para este problema. Este operador tiene como objetivo incrementar la diversidad de la asignación de modo a la vez que busca la factibilización de las soluciones que luego de pasar por los operadores genéticos superan la disponibilidad de los recursos no renovables del proyecto.

El algoritmo genético híbrido que se propone para resolver el MRCPSP incorpora todos los elementos descritos anteriormente y al ser comparado con los métodos propuestos en la literatura se concluye que tiene un desempeño comparable con los mejores métodos desarrollados, tanto en la calidad de las soluciones obtenidas como en el tiempo de cómputo necesitado para calcularlas.

6.1.3. Generación de Programaciones Robustas

El problema de la generación de programaciones surge al reconocer el impacto que tiene la incertidumbre en la ejecución del proyecto. Dicha incertidumbre puede ser tratada proactivamente en la fase de programación o reactivamente en la fase de ejecución.

Nuestro enfoque, aunque se centra en el desarrollo de un método para generar programaciones estables de manera proactiva, reconoce que dicha programación no será capaz de hacer frente a todas las eventualidades y por ello usa un método reactivo en la fase de ejecución.

El objetivo en este problema es minimizar la variación entre el tiempo de inicio esperado de las actividades y el tiempo de inicio realizado. Asumimos que si una programación es robusta, entonces la variación será cercana a cero.

Para abordarlo hemos diseñados un algoritmo genético que actúa en dos aspectos de manera paralela. Por una parte inserta buffers para aumentar la estabilidad de la programación, a la vez prueba diferentes secuencias de actividades, permitiendo así explorar las diversas combinaciones que se pueden generar.

Debido a que no es posible conocer de antemano la robustez de la programación generada y dicha robustez es el objetivo del problema, hemos implementado una función de fitness que utiliza como medida surrogada el impacto que tiene el buffer insertado en la programación según la importancia de la actividad en la estabilidad de la misma.

Para determinar la importancia de la actividad en la estabilidad de la programación hemos probado diferentes conjuntos de pesos de ponderación relacionados con características de la actividad tales como su duración o su uso de recursos.

Para medir el desempeño de las programaciones así generadas, se implementa un simulador que introduce variabilidad en la duración de las actividades. La robustez de la solución generada se mide de dos maneras diferentes. En primera instancia se mide el retraso porcentual del proyecto. En segundo lugar se mide la desviación media de las actividades.

Debido a que no existen librerías estándar de comparación para este problema y a la aleatoreidad que existe en el proceso de simulación es imposible comparar nuestro trabajo con los publicados previamente. Por ello hemos diseñado un algoritmo que asigna de manera aleatoria los buffers en la programación y lo hemos utilizado como método de referencia. El Algoritmo Genético diseñado obtiene soluciones que se comportan de manera más estable en los escenarios de simulación.

En cuanto al estudio de las características de las actividades determinamos que la duración es la característica con mayor impacto en el retraso total del proyecto. En cuanto al retraso promedio de las actividades, al utilizar el criterio de Número Total de Predecesores se obtienen los mejores resultados, es decir mayor estabilidad en el inicio de todas las actividades de la programación

6.2. Líneas Futuras de Investigación

El estudio llevado a cabo en esta tesis en el caso del RCPSP y MRCPSP se ha centrado en la resolución del objetivo de minimizar la duración del proyecto. Los algoritmos desarrollados pueden adaptarse para estudiar otros objetivos diferentes, por ejemplo la ociosidad de los recursos, la nivelación de su demanda. Así mismo pueden ampliarse para el estudio de otras versiones del problema, como cuando son posibles las interrupciones de las actividades o con otro tipo de relaciones de precedencia.

Otra área de trabajo es el desarrollo de algoritmos multiobjetivo, donde además de minimizar la duración del proyecto se permita incluir otro tipo de criterios. En el RCPSP podría ser la nivelación del uso de recursos, que facilita la gerencia del proyecto.

Debido a los buenos resultados obtenidos, junto con los cortos tiempos de ejecución requeridos, hace que los métodos desarrollados sean adecuados para su incorporación a software profesional de gestión de proyectos a través de los lenguajes de programación que estos paquetes facilitan, siendo este aspecto una posible dirección de trabajo en el futuro próximo.

En cuanto a la generación de programaciones robustas el problema MRCPSP tiene un gran interés como línea de trabajo, tanto en el desarrollo de procedimientos proactivos como reactivos ya se puede explotar la posibilidad "on-line" de cambiar el modo de ejecución de las actividades para mantener la programación.

Finalmente, una línea de investigación intersante para el grupo de investigación en el cual se hace esta tesis doctoral es la adaptación de los algoritmos desarrollados para la resolución de otros problemas de scheduling como el de la programación de horarios ferroviarios (OPT).

6.3. Publicaciones

A continuación se presentan las publicaciones relacionadas con el trabajo de investigación realizado durante esta tesis doctoral, que comprenden artículos en revistas internacionales y comunicaciones a congresos.

Publicaciones en Revistas Internacionales

1. **An efficient hybrid genetic algorithm for scheduling projects with resource constraints and multiple execution modes**. A. Lova, P. Tormos, M. Cervantes, F. Barber. *International Journal of Production Economics*. Volume 117, Issue 2, Febrero 2009, Pg 302-316 (SCI/JCR 2.026)

   Este artículo se presenta el diseño de un algoritmo genético para la solución del MRCPSP. Una de las principales aportaciones es el diseño

de un método de mejora que permite aprovechar las holguras libres de las actividades para encontrar modos más eficientes que reduzcan la duración de la secuencia.
Otras contribuciones importantes son el método de asignación de modos de la población inicial, que de manera muy eficiente permite generar individuos factibles y la función objetivo la cual permite penalizar a los individuos no factibles en proporción al exceso de recursos consumidos. Un extensivo estudio computacional demuestra que es competitivo si se compara con los métodos disponibles actualmente

2. **Experimental analysis of optimization techniques on the road passenger transportation problem**. B. Lopez, V. Muñoz, J. Murillo, F. Barber, M.A. Salido, M. Abril, M. Cervantes. *Engineering Applications of Artificial Intelligence*. Volume 22, Issue 3, Abril 2009, Pg 374-388 (SCI/JCR 1.397)

   Este artículo es resultado de un trabajo conjunto con el grupo de Ingeniería de Control y Sistemas Inteligentes de la Universidad de Gerona.
   En este artículo se comparan métodos exactos y aproximados de resolución para un problema de transporte urbano. Los métodos metaheurísticos desarrollados son capaces de solucionar todo el conjunto de problemas de manera satisfactoria, obteniendo soluciones cercanas al óptimo. Métodos más tradicionales como los CSP sólo son capaces de resolver problemas de tamaño pequeño.

3. **A genetic approach to robust project scheduling**. M. Cervantes, A. Lova, P. Tormos, F. Barber. Aceptado para Revisión en *International Journal of Production Economics* (SCI/JCR 2.026)

   En este trabajo se aborda el problema de la generación de secuencias robustas y de los métodos reactivos que deben emplearse una vez ejecutado el proyecto. Como método proactivo se propone un algoritmo genético que inserta buffers en diversas actividades. Se estudian diversas características de las actividades como factor determinante del impacto en la estabilidad de la programación.
   Para la evaluación de los individuos se diseña una función basada en la holgura que se genera entre las actividades luego de insertar los tiempos de seguridad así como en el impacto de la actividad en la estabilidad de la programación.
   Se diseñan dos métricas para evaluar la robustez de las programaciones luego de someterlas a un proceso de simulación de la ejecución en escenarios de alta y baja variabilidad de las actividades. Para evaluar la efectividad del Algoritmo Genético se compara con un método aleatorio de inserción de buffers.

Comunicaciones a Congresos

1. **A Hybrid Genetic Algorithm for the Multi-Mode Resource Constrained Project Scheduling Problem**. A. Lova, P. Tormos, M. Cervantes, F. Barber. *XI International workshop on Project management and Scheduling-PMS*. Estambul Turquía 2008. Pg 193-196

   El PMS es un congresos de alta relevancia científica en el área de programación de proyectos. En este trabajo se introduce el método de mejora FBI extendido al problema multimodo. Este método permite el cambio de modo de las actividades cada vez que se hace una pasada forward o backward si dicho cambio permite disminuir el tiempo de finalización de la actividad en la secuencia parcial actual.
   Así mismo se calcula el esfuerzo computacional de dicho método de mejora el cual es necesario para contabilizar las secuencias generadas, que es el parámetro que permite la comparación entre los algoritmos desarrollados.

2. **Dynamic Population Steady-State Genetic Algorithm for the Resource-Constrained Project Scheduling Problem**. M. Cervantes, A. Lova, P. Tormos, F. Barber. *XXI International Conference on Industrial Engineering and Other Applications of Applied Intelligent Systems IEA/AIE*. Wroclaw Polonia 2008. *LNCS Vol 5027* Pg 611-620.

   Este artículo presenta el diseño de un algoritmo genético para el problema RCPSP. Introduce dos novedades con respecto a Algoritmos Genéticos previos: por un lado utiliza una población variable que permite la diversificación de la población. Por otra parte utiliza la sustitución inmediata como mecanismo para aumentar la presión selectiva. Utiliza el método de mejora FBI de manera selectiva aplicándolo sólo a aquellas secuencias que son más prometedoras en cada instante. El estudio computacional muestra que es competitivo frente a los métodos actuales.
3. **An efficient Adaptive Heuristic for the resource constrained Project scheduling problem**. A. Lova, P. Tormos, M. Cervantes, F. Barber. *XII Conferencia de la Asociación Española para la Inteligencia Artificial-CAEPIA*. Salamanca, España. 2007 Volumen II, Pg 259-269

   Este trabajo presenta el resultado de diversos métodos heurísticos de una sola pasada basados en reglas de prioridad y en la aplicación del procedimiento de mejora FBI.
   El análisis de los resultados de las heurísticas de una sola pasada permitió determinar las reglas de prioridad más efectivas y en una segunda etapa se diseñó una heurística multipasada que aplica el método FBI de manera selectiva. El estudio computacional mostró que dicho método heurístico obtiene resultados similares a algunos métodos metaheurísticos reportados en la literatura.

Referencias

1. J. Alcaraz and C. Maroto. A robust genetic algorithm for resource allocation in project scheduling. *Annals of Operations Research*, 102:83–109, 2001.
2. J. Alcaraz and C. Maroto. A hybrid genetic algorithm based on intelligent encoding for project scheduling. In J. Józefowska and J. Weglarz, editors, *Perspectives in modern project scheduling*, pages 249–274. Springer-Verlag, 2006.
3. J. Alcaraz, C. Maroto, and R. Ruiz. Solving the multi-mode resource-constrained project scheduling problem with genetic algorithms. *Journal of Operational Research*, 54:614–626, 2003.
4. J. Alcaraz, C. Maroto, and R. Ruiz. Improving the performance of genetic algorithms for the rcpsp problem. In *Proceedings of the Ninth International Workshop on Project Management and Scheduling*, pages 40 – 43, 2004.
5. M. Aloulou and M.Portmann. An efficient proactive reactive scheduling approach to hedge against shop floor disturbances. In *Proceedings of MISTA 2003*, 2003.
6. R. Alvarez-Valdes and J.M. Tamarit. The project scheduling polyhedron:dimension, facets and lifting theorems. *European Journal of Operational Research*, 67:204–220, 1993.
7. C. Artigues and F.Roubellat. A polynomial activity insertion algorithm in a multi-resource schedule with cumulative constraints and multiple modes. *European Journal of Operational Research*, 127(2):297 – 316, 2000.
8. H. Aytug, M.A. Lawley, K. Mckay, S. Mohan, and R. Uzsoy. Executing production schedules in the face of uncertainties: A review and some future directions. *European Journal of Operational Research*, 161:86–110, 2005.
9. T. Baar, P. Brucker, and S. Knust. Tabu-search algorithms for the resource-constrained project scheduling problem. In S. Voss, S. Martelo, I. Osman, and C Roucairol, editors, *Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization*, pages 1–18. Kluwer Academic, 1999.
10. J. Blazewicz, H.K. Ecker, E. Pesch, G. Schmith, and J. Werlagz. *Scheduling Computer and Manufacturing Processes*. Springer-Verlag, 2001.
11. J. Blazewicz, J.K. Lenstra, and A.H.G. Rinooy Kan. Scheduling subject to resource constraints: Classification and complexity. *Discrete Applyed Math*, 5:11–24, 1983.

12. F. F. Boctor. Heuristics for shceduling projects with resource restrictions and several resource-duration modes. *International Journal of Production Research*, 31:2547–2558, 1993.
13. F. F. Boctor. A new and efficient heuristic for scheduling projects with resource restrictions and multiple execution modes. *European Journal of Operational Research*, 90:349–361, 1996.
14. F. F. Boctor. Resource-constrained project scheduling by simulated annealing. *International Journal of Production Research*, 34:2335–2351, 1996.
15. K. Bouleimen and H Lecocq. A new efficient simulated annealing algorithm for the resource-constrained project scheduling problem and its multiple modes version. *European Journal of Operational Research*, 149:268–281, 2003.
16. P. Brucker, A. Drexl, R. Möhring, K. Neumann, and E. Pesch. Resource-constrained project scheduling: Notation, classification, models and methods. *European Journal of Operational Research*, 112:3–41, 1999.
17. P. Brucker, S. Kunst, A. Schoo, and O. Thierel. A branch and bound algorithm for the resource constrained project scheduling. *European Journal of Operational Research*, 107:272–288, 1998.
18. M. Cervantes, A. Lova, P. Tormos, and F. Barber. A dynamic population steady-state genetic algorithm for the resource-constrained project scheduling problem. *Lecture Notes in Artificial Intelligence*, 5027.
19. A. Cesta, A. Oddi, and S.F. Smith. Profile based algorithms to solve multiple capacitated metric scheduling problems. In *AIPS-98*, 1998.
20. N. Christofides, R. Álvarez Valdés, and J.M. Tamarit. Project scheduling with resource constraints: A branch and bound approach. *European Journal of Operational Research*, 29:262–273, 1987.
21. J. Coelho and L. Tavares. Comparative analysis of metaheuricstics for the resource constrained project scheduling problem. Technical report, Department of Civil Engineering, Instituto Superior Tecnico, Portugal, 2003.
22. L. Davis. Job shop scheduling with genetic algorithms. In *Proceedings of the First International Conference on Genetic Algorithms and their applications*, 1985.
23. D. Debels, B. De Reyck, and M. Vanhoucke R. Leus. A hybrid scatter search/electromagnetism meta-heuristic for project scheduling. *European Journal of Operational Research*, 169:638 – 653, 2006.
24. D. Debels and M. Vanhoucke. A bi-population genetic algorithm for the resource constrained project scheduling problem. *LNCS*, 3483, 2005.
25. D. Debels and M. Vanhoucke. A decomposition-based genetic algorithm for the resource-constrained project scheduling problem. *Operations Research*, 55:454–469, 2007.
26. E.L. Demeulemeester and W. Herroelen. A branch-and-bound procedure for the multiple resource-constrined project scheduling problem. *Management Science*, 98:1803–1818.
27. E.L. Demeulemeester and W. Herroelen. *Project Scheduling: A Research Handbook.* Kluwer Academic Publishers, 2002.
28. A. Drexl. Scheduling of project networks by job assignment. *Manage. Sci.*, 37.
29. S.E. Elmaghraby. *Activity Networks: Project Planning and Control by Network Models.* Wiley, N.Y., 1977.
30. F.. Glover. Future paths for integer programming and links to artificial intelligence. *Computers and Operations Research*, 13:533–549, 1986.

31. F. Glover and M Laguna. *Tabu Search.* Kluwer Academic Publishers, 1997.
32. D. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison-Wesley, 1989.
33. D. Goldberg and R. Lingle. Alleles, loci, and the traveling salesman problem. In *Proceedings of the First International Conference on Genetic Algorithms and Their Applications*, 1985.
34. E. Goldratt. *Critical Chain.* North River Press.
35. S. Hartmann. Competitive genetic algorithm for resource-constrained project scheduling. *Naval Research Logistics*, 45, 1998.
36. S. Hartmann. Project Scheduling under limited resources - Models, Methods and Applications. volume 418 of *Lecture Notes in Economics and Mathematical Systems*. Springer Germany, 1999.
37. S. Hartmann. Project scheduling with multiple mode: a genetic algorithm. *Annals of Operations Research*, 102, 2001.
38. S. Hartmann. A self-adapting genetic algorithm for project scheduling under resource constraints. *Naval Research Logistics*, 49, 2002.
39. S. Hartmann and A. Drexl. Project scheduling with multiple modes: A comparison of exact algorithms.
40. S. Hartmann and R. Kolisch. Experimental evaluation of state-of-the-art heuristics for the resource-constrained project scheduling. *European Journal of Operational Research*, 127, 2000.
41. W. Herroelen and R. Leus. On the merits and pitfalls of critical chain scheduling. *Journal of Operations Management*, 19(5):559 – 577, 2001.
42. W. Herroelen and R. Leus. The construction of stable project baseline schedules. *European Journal of Operational Research*, 156(3):550 – 565, 2004.
43. W. Herroelen and R. Leus. Robust and reactive project scheduling: a review and classification of procedures. *International Journal of Production Research*, 42:1599–1620(22), 2004.
44. W. Herroelen and R. Leus. Project scheduling under uncertainty: Survey and research potentials. *European Journal of Operational Research*, 165(2):289 – 306, 2005.
45. G. Igelmund and F. G. Radermacher. Preselective strategies for the optimization of stochastic project networkds under resource constraints. *Networks*, 13:1–28, 1983.
46. J. Jozefowska, M. Mika, R. Rozycki, G. Waligora, and J. Weglarz. Simulated annealing for multi-mode resource-constrained project scheduling. *Annals of Operations Research*, 102:137–155, 2001.
47. L.A. Kaplan. *Resource-Constrained Project Schedulling with Preemption of Jobs.* PhD thesis, University of Michigan, 1988.
48. S. Kirkpatrick, C.D. Gelatt, and M.P. Vecchi. Optimization by simulated annealing. *Science*, 220:671–680, 1983.
49. R. Klein. Biderectional planning: Improving priority rule based heuristics for scheduling resource-constrained projects. *European Journal of Operational Research*, 127:619–638, 2000.
50. R. Klein. Project scheduling with time-varying resource constraints. *International Journal of Production Research*, 38:3937–3952, 2000.
51. Y. Kochetov and A. Stolyar. Evolutionary local search with variable neighborhood for the resource constrained project scheduling problem. In *Proceedings of the 3rd International Workshop of Computer Science and Information Technologies, 2003*, 2003.

52. R. Kolisch. *Project scheduling under resource constraints : Efficient heuristics for several problem classes*. Heidelberg: Physica-Verlag, 1995.
53. R. Kolisch. Serial and parallel resource constrained project scheduling methods revisited: Theory and computation. *European Journal of Operational Research*, 90:320–333, 1995.
54. R. Kolisch and A. Drexl. Adaptive search for solving hard project scheduling problems. *Naval Research Logistics*, 43:23–40, 1996.
55. R. Kolisch and A. Drexl. Local search for nonpreemptive multi-mode resource constrained project scheduling. *IIE transactions*, 29:987–999, 1997.
56. R. Kolisch and S. Hartmann. Heuristic algorithms for solving the resource-constrained project scheduling: Classification and computational analysis. In Jan Weglarz, editor, *Handbook of recent Advances in Project Scheduling*. Kluwer Academic Publishers, 1999.
57. R. Kolisch and S. Hartmann. Experimental investigation of heuristics for resource- constrained project scheduling: An update. *European Journal of Operational Research*, 174:23–37, 2006.
58. R. Kolisch and R. Padman. An integrated survey of deterministic project scheduling. *OMEGA*, 29, 2001.
59. R. Kolisch and A. Sprecher. Psplib - a project scheduling library. *European Journal of Operational Research*, 96, 1996.
60. R. Kolisch, A. Sprecher, and A. Drexl. Characterization and generation of a general class of resource-constrained project scheduling problems. *Management Science*, 41, 1995.
61. O. Lambrechts, Demeulemeester E, and Herroelen W. Proactive and reactive strategies for resource-constrained project scheduling with uncertain resource availabilities. *Journal of scheduling*, 11(2):121 – 136, 2008.
62. O. Lambrechts, Demeulemeester E, and Herroelen W. A tabu search procedure for developing robust predictive project schedules. *International Journal of Production Economics*, 111(2):493 – 508, 2008. Special Section on Sustainable Supply Chain.
63. J. Lancaster and M. Ozbayrak. Evolutionary algorithms applied to project scheduling problems: A survey of the state of the art. 45:425–450, 2007.
64. V.J. Leon and B. Ramamoorthy. Strength and adaptability of problem-space based neighborhoods for resource-constrained scheduling. *OR Spektrum*, 17:173–182, 1995.
65. R. Leus and W. Herroelen. The complexity of machinescheduling for stability with a single disrupted job. *IIE Transactions*, 26:32–43, 2005.
66. F.G. Lobo and C.F. Lima. A review of adaptive population sizing schemes in genetic algorithms. In *GECCO '05: Proceedings of the 2005 workshops on Genetic and evolutionary computation*, pages 228–234, 2005.
67. A. Lova, , P. Tormos, and F. Barber. Multi-mode resource constrained project scheduling: Scheduling schemes, priority rules and mode selection rules. *Revista Iberoamericana de Inteligencia Artificial*, 30:69–86, 2006.
68. A. Lova, , P. Tormos, M. Cervantes, and F. Barber. An efficient hybrid genetic algorithm for scheduling projects with resource constraints and multiple execution modes. *International Journal of Production Economics*, 117:302–316, 2009.
69. A. Lova, P. Tormos, M. Cervantes, and F. Barber. An efficient adaptive heuristic for the resource constrained project scheduling problem. In *Proceedings of CAEPIA-TIA 2007*, Salamanca, Spain, 2007.

70. B. Melián, J.A. Moreno, and Moreno J.M. Metaheuristics a global view. *Revista Iberoamericana de Inteligencia Artificial*, 19:7–28, 2003.
71. J.J.M. Mendes, J.F. Goncalves, and M. G. C. Resende. A random key based genetic algorithm for the resource constrained project scheduling problem. *Computers and Operations Research*, 36:92–109.
72. A. Mingozzi, V. Maniezzo, S. Ricciadelli, and L. Bianco. An exact algorithm for the resource-constrained project schedulingproblem based on a new mathematical formulation. *Management Science*, 44:715–729, 1998.
73. M. Mori and C.C. Tsheng. A genetic algorithm for multi-mode resource constrained project scheduling problem. *European Journal of Operational Research*, 100:164–141, 1997.
74. K. Nonobe and T. Ibaraki. Formulation and tabu search algorithm for the resource constrained project scheduling problem. In C. C. Ribeiro and P. Hansen, editors, *Essays and Surveys in Meta-heuristics*, pages 557–588. Kluwer Academic Publishers, 2002.
75. L. Ozdamar. A genetic algorithm approach to a general category project scheduling problem. *IEEE Trans. on Systems, Man and Cybernetics*, 29(1):44–59, 1999.
76. J.H. Patterson, R. Slowinski, F.B Talbot, and J. Weglarz. Algorithm for a general class of precedence and resource constrained scheduling problems. In R. Slowinski and J. Weglarz, editors, *Advances in Project Scheduling*, pages 3–28. 1989.
77. Michael Pinedo. *Planning and Scheduling in Manufacturing and Services*. Springer, 2000.
78. E. Pinson, C. Prins, and F. Rullier. Using tabu search for solving the resource-constrained project scheduling problem. In *International Workshop on Project Management and Scheduling, PMS'94*, pages 102–106, Louvain, Belgique, July 1994.
79. Project Management Insititue. PMI. *A Guide to the Project Management Body of Knowledge: (PMBOK® Guide)*. Project Management Institute, 2004.
80. N. Policella, S.F. Smith, A. Cesta, and A. Oddi. Generating robust schedules through temporal flexibility. In *4th International Conference on Automated Planning & Scheduling, ICAPS04*, pages 209–218, 2004.
81. A.B. Pritsker and P.M. Watters, L.J.and Wolfe. Multiproject scheduling with limited resources: A zero-one programming approach. *Management Science*, 16:93–108, 1969.
82. M. Ranjbar, B. De Reyck, and F. Kianfar. A hybird scatter search for the discrete time/resource trade-off problem in project scheduling. *European Journal of Operational Research*, 119:35–48, 2009.
83. C. Romero. *Técnicas de Programación y Control de Proyectos*. Piramide, 2002.
84. A. Schirmer. Case-based reasoning and improved adaptive search for project scheduling. *Naval Research Logistics*, 47:201–222, 2000.
85. C. Schwindt. *Resource Allocation in Project Management*. GOR Publications. Springer, 2005.
86. R. Slowinski, B. Soniewicki, and J. Weglarz. Dss for multiobjective project scheduling. *European Journal of Operational Research*, 79:220–229, 1994.
87. A. Sprecher. Resource-constrained project scheduling - exact methods for the multi-mode case. Lecture Notes in Economics, No. 409, 1994.

88. A. Sprecher and A. Drexl. Multi-mode rsource-constrained project scheduling by a simple, general and powerful sequencing algorithm. *European Journal of Operational Research*, 107, publisher=.
89. A. Sprecher, S. Hartmann, and A. Drexl. An exact algorithm for project scheduling with multiple modes. *OR Spektrum*, 19:195–203, 1997.
90. J.P. Stinson, E.W. Davis, and B.M. Khumawala. Multiple resource-constrained scheduling using branch and bound. *AIIE Transactions*, 10:252–259, 1978.
91. R. Stork. *Stochastic Resource Constrained Project Scheduling*. PhD thesis, School of Mathematics and Natural Sciences, Technical University of Berlin, 2001.
92. F.B. Talbot. Resource-constrained project scheduling problem with time-resource trade-offs: The non preemptive case. *Management Science*, 28:1197–1210, 1982.
93. P. R. Thomas and S. Salhi. A tabu search approach for the resource constrained project scheduling problem. *Journal of Heuristics*, 4:123–139, 1998.
94. P. Tormos and A. Lova. A competitive heuristic solution technique for resource-constrained project scheduling. *Annals of Operations Research*, 102:65–81, 2001.
95. P. Tormos and A. Lova. An efficient multi-pass heuristic for project scheduling with constrained resources. *International Journal of Production Research*, 41:1071–1086, 2003.
96. P. Tormos and A. Lova. Integrating heuristics for rcpsp: One step forward. Technical report, Universidad Politécnica de Valencia, 2003.
97. V. Valls, F. Ballestin, and S. Quintanilla. A hybrid genetic algorithm for the resource constrained project scheduling problem with the peak crossover operator. In *Eighth International Workshop on Project Management and Scheduling*, pages 368–371, 2002.
98. V. Valls, Ballestin F., and Quintanilla M.S. A hybrid genetic algorithm for the rcpsp. Technical report, Universidad de Valencia, 2003.
99. V. Valls, Ballestin F., and Quintanilla M.S. Justification and rcpsp: A technique that pays. *European Journal of Operational Research*, 165:372–386, 2005.
100. S. Van De Vonder, E. Ballestín, F. Demeulemeester, and W. Herroelen. Heuristic procedures for reactive project scheduling. *Computers and Industrial Engineering*, 52(1):11 – 28, 2007.
101. S. Van De Vonder, E. Demeulemeester, and W. Herroelen. Proactive heuristic procedures for robust project scheduling: An experimental analysis. *European Journal of Operational Research*, 189(3):723 – 733, 2008.
102. S. Van De Vonder, E. Demeulemeester, W. Herroelen, and R. Leus. The use of buffers in project management: The trade-off between stability and makespan. *International Journal of Production Economics*, 97(2):227 – 240, 2005.
103. S. Van De Vonder, E. Demeulemeester, W. Herroelen, and R. Leus. The trade-off between stability and makespan in resource-constrained project scheduling. *International Journal of Production Research*, 44(2):215 – 236, 2006.
104. V. Van Petghem and M. Vanhoucke. A genetic algorithm for the multi-mode resource-constrained project scheduling problem. Technical report, Faculty of Economics, Ghent University, 2008.
105. M. Wall. *A genetic algorithm for resource-constrained scheduling*. PhD thesis, Massachusetts Institute of Technology, 1996.

106. H. Zhang, H. Li, and C.M. Tam. Multimode project scheduling based on particle swarm sptimization. *Computer Aided Civil and Infraestructure Engineering*, 21:93–103, 2006.