Boletín de Estadística e Investigación Operativa
Vol. 25, No. 1, Febrero 2009, pp. 22-31

INVESTIGACIÓN OPERATIVA

Genetic Algorithms for the Resource-Constrained Project Scheduling Problem

Javier Alcaraz Soria
Centro de Investigación Operativa, Departamento de Estadística, Matemáticas e Informática
Universidad Miguel Hernández de Elche
✉ jalcaraz@umh.es

Concepción Maroto Álvarez
Departamento de Estadística e Investigación Operativa Aplicadas y Calidad
Universidad Politécnica de Valencia
cmaroto@eio.upv.es

Abstract

Genetic Algorithms have been applied to many different optimization problems and they are one of the most promising metaheuristics. In the last years we have focused our work developing efficient genetic algorithms to solve the Resource-Constrained Project Scheduling Problem. We have proposed several algorithms to solve different versions of the problem, designing new representations for the solutions, different crossover techniques, innovative mutation mechanisms, efficient local search procedures and other features that will be summarized in this work.

Keywords: Project Scheduling, Genetic Algorithms, Metaheuristics, Resource Allocation.
AMS Subject classifications: 90B35, 90B99, 90-08.

1. Introducción: El problema

El problema de la Programación de Proyectos con Recursos Limitados (PPRL), al que se hace referencia en la literatura como RCPSP (*Resource-Constrained Project Scheduling Problem*), es aquel cuyo objetivo es minimizar la duración del proyecto. Las actividades que lo componen no pueden interrumpir su ejecución y están sujetas exclusivamente a relaciones de precedencia de tipo Fin-Inicio. Además, utilizan sólo recursos renovables con disponibilidad limitada y constante a lo largo de todo el proyecto y presentan un único modo

de ejecución, es decir cada actividad se puede ejecutar de una única forma (con una duración determinada y un consumo dado de recursos).

A partir de aquí, haremos referencia a este problema simplemente como el problema de la programación de proyectos con recursos limitados, aunque también se suele hacer referencia a él como la versión estándar del problema o la versión "único-modo". Se han propuesto dos clasificaciones y notaciones diferentes para este tipo de problemas. Éstas son las propuestas por Brucker et al. [1] y Herroelen et al. [2]. En ambas, el problema queda definido mediante tres campos: el primero de ellos hace referencia a los recursos utilizados, el siguiente a las características de las actividades y el tercero al criterio de optimalidad o función objetivo.

Según la notación propuesta por Brucker et al. la versión estándar del problema vendría representada como *PS/prec/Cmax*, donde *PS* indica (*project scheduling*) programación de proyectos, para distinguirlo de los problemas de máquinas o *"job-shop"*, *prec* indica que entre las actividades existen sólo relaciones de precedencia de tipo general y *Cmax* hace referencia a la función objetivo, indicando que se intenta minimizar el instante máximo de finalización de las actividades, es decir, se intenta minimizar la duración del proyecto.

Según la clasificación propuesta por Herroelen et al. el problema quedaría definido como *m,1/cpm/Cmax*. La interpretación es la siguiente: el primer parámetro indica que se utiliza un número *m*, no determinado a priori, de tipos de recursos y *1* indica que sólo se utilizan recursos renovables. El siguiente campo hace referencia a las características de las actividades, y en concreto *cpm* indica que entre las actividades existen únicamente relaciones de precedencia de tipo fin-inicio sin retraso, es decir, sin solapamientos ni desplazamientos. El tercer campo, igual que en la primera representación, indica que el objetivo es minimizar la duración del proyecto.

Este problema presenta distintas variantes, una de las cuales consiste en que 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. Por ejemplo, la tarea de realizar el análisis de requerimientos para una aplicación informática puede realizarse por un solo analista en dos semanas, o por un analista y un programador en una semana y media. A esta versión se hace referencia como la versión "multi-modo"del problema (MRCPSP en inglés y PPRL-MM en castellano), en la que pueden aparecer además recursos no renovables (fungibles) y doblemente restringidos (aquellos que son fungibles y para los que se tiene además un consumo máximo permitido por unidad de tiempo).

Además, 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 (renovables, no renovables y doblemente restringidos), etc. Además se pueden establecer objetivos diferentes al de minimizar la duración del proyecto, como por ejemplo maximizar el valor neto del proyecto.

En este trabajo se presentan los diferentes diseños y desarrollos que hemos llevado a cabo para las versiones del problema estándar y multi-modo.

2. Algoritmos genéticos para la versión estándar del problema

En este apartado presentamos los diferentes desarrollos llevados a cabo para el diseño de un algoritmo genético eficiente y robusto para resolver el problema en su versión estándar. Presentamos en primer lugar distintos tipos de codificación para las soluciones, diferentes técnicas de cruce y mutación, así como técnicas para mejorar la calidad de las soluciones.

2.1. Codificación de las soluciones

La codificación de las soluciones del problema es un aspecto crucial en el comportamiento del algoritmo, y además, de dicha codificación dependerá el diseño del resto del algoritmo. Por tanto, la elección de una codificación apropiada es la primera fase del diseño y posiblemente la más importante, porque pueden existir grandes diferencias entre los resultados ofrecidos por un algoritmo que utilice una determinada codificación u otra.

La codificación más utilizada en algoritmos metaheurísticos para resolver el problema que nos ocupa es la llamada **codificación por lista de actividades**. En ella, cada individuo de la población se representa mediante un cromosoma con tantos genes como actividades tiene el proyecto. Además, esta lista es una lista ordenada, de forma que una determinada actividad siempre aparece detrás de todas sus predecesoras. Para transformar esta secuencia de actividades en un calendario de ejecución se aplica el método de secuenciación basado en el esquema serie. Este algoritmo se conoce actualmente como algoritmo serie hacia delante (*forward*), ya que posteriormente se ha desarrollado también una versión hacia atrás (*backward*). En su versión hacia delante, el procedimiento consiste en secuenciar cada actividad en su fecha factible más temprana. Es decir, en el primer instante en el que existe disponibilidad de recursos para poder ejecutar sin interrupciones la actividad, a partir del instante en el que finalizan todas sus precedentes.

Nuestra primera propuesta para codificar las soluciones se basa en la representación estándar por lista de actividades descrita antes. Ahora una solución vendrá representada igualmente por una lista ordenada de actividades, pero a la que se ha añadido un gen o bit, que denominaremos bit modo de secuenciación y

puede tomar únicamente dos valores: *forward* o *backward*. Por simplicidad también le llamaremos gen o bit *forward/backward* o *f/b*. Dicho gen indica el modo de secuenciación empleado para construir el calendario correspondiente, es decir para construir dicho calendario aplicaremos el esquema serie, en su versión hacia delante o hacia atrás, dependiendo de lo que indique el bit añadido. Con este nuevo tipo de representación, no sólo la solución al problema está sujeta a optimización, sino también la estructura algorítmica utilizada para construir dicha solución. La secuenciación serie hacia atrás es muy similar a la versión hacia delante. La única diferencia es que ahora las actividades se secuencian en orden inverso. Si en la versión *forward*, la actividad inicio del proyecto es la primera en ser secuenciada, en el proceso *backward* será la última. Es decir, se comienza secuenciando la última de la lista y se sigue por orden hasta acabar secuenciando la primera. Ahora una actividad se podrá secuenciar cuando todas sus sucesoras lo hayan sido. Una actividad se secuenciará lo más tarde posible, es decir, en el instante más tardío posible de forma que haya recursos suficientes para su ejecución y ésta no acabe después de que empiece cualquiera de sus sucesoras. Aplicando el modo *forward* se pueden construir programaciones o calendarios que no podrían ser obtenidos a través del modo *backward* y viceversa. Más adelante, cuando describamos el mecanismo de cruce diseñado para este tipo de representación, veremos como es explotada eficientemente esta característica. Como en la representación por lista de actividades, una solución representa un único calendario, pero diferentes soluciones se pueden transformar en el mismo calendario de ejecución.

Aunque la codificación anterior ofreció excelentes resultados en los diferentes estudios computacionales llevados a cabo, refinamos y completamos dicha codificación añadiendo todavía más información específica del problema. En concreto se añadió otro gen más, el gen serie/paralelo (en adelante gen *s/p*) que indica si se utiliza el esquema de secuenciación serie o paralelo para construir el calendario de ejecución del proyecto. Este gen ya había sido añadido por Hartmann [3] a la representación estándar por lista de actividades con excelentes resultados. Sin embargo, ahora la combinación del bit *f/b* con el bit *s/p* permite construir el calendario de ejecución, a partir de la lista de actividades, en base a cuatro esquemas diferentes: serie-*forward*, serie-*backward*, paralelo-*forward* y paralelo-*backward*. La nueva codificación puede observarse en la Figura 1. Una descripción más detallada de estos tipos de codificación puede consultarse en [4] y [5].

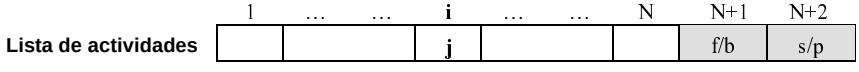

Figura 1: Nueva codificación, versión estándar

2.2. Cruce

Aunque se han desarrollado diferentes mecanismos de cruce aplicados a diferentes codificaciones, el que mejores resultados ha obtenido es el proceso de cruce que opera sobre la codificación que incorpora los bits *f/b* y *s/p*, puesto que explota de forma eficiente la información contenida en dicha representación. El esquema básico del operador se ha representado en la Figura 2.

Se trata de una versión de cruce dos puntos, que parte a cada una de las soluciones en tres partes. La operación de cruce depende, además de los dos puntos de cruce, del bit *f/b* de los padres. El bit *s/p* no influye en el mecanismo y cada hijo lo hereda directamente de uno de los padres. El bit *f/b* de cada padre decide si el hijo correspondiente hereda los genes de izquierda a derecha o al contrario. En definitiva, se tienen 16 posibles parejas diferentes en cada operación de cruce. Los experimentos llevados a cabo han demostrado que se trata de un cruce muy eficiente que utiliza de forma adecuada la información específica del problema que contiene cada individuo. Una descripción más detallada de este mecanismo de cruce puede consultarse en [5].

2.3. Mutación

El mecanismo de mutación imita el proceso aleatorio de mutación que sufre el ADN de las especies. Dicha mutación permite dotar al individuo de ciertas características, buenas o no, que quizás no existan en la población actual. Si estas nuevas aptitudes son beneficiosas, seguramente pasarán de generación en generación y cada vez aparecerán en más individuos de la población. Si no lo son, probablemente desaparecerán con la evolución. Nuestra aportación en el mecanismo de mutación fue adaptar como tal un procedimiento utilizado por Boctor [6] en un algoritmo de Simulated Annealing para generar el vecindario de un individuo dado. En primer lugar, el mecanismo de mutación actúa sobre la lista de actividades, de forma que cada actividad puede ser desplazada a cualquier posición aleatoria, con una determinada probabilidad de mutación, siempre y cuando esta nueva posición satisfaga las relaciones de precedencia. Es decir, la mutación no puede generar soluciones no factibles. Se diseñaron diversos mecanismos de mutación basados en esta idea, que se diferenciaban fundamentalmente en el hecho de alterar también o no los bits *f/b* y *s/p* de las soluciones. Tras diversas pruebas, finalmente se optó por un procedimiento que en una segunda fase pudiera alterar también dichos genes, lo que le daba al procedimiento mayor poder de diversificación. En [5] se describe con detalle el procedimiento y se compara con otros mecanismos de mutación propuestos por otros autores, demostrando su buen comportamiento.

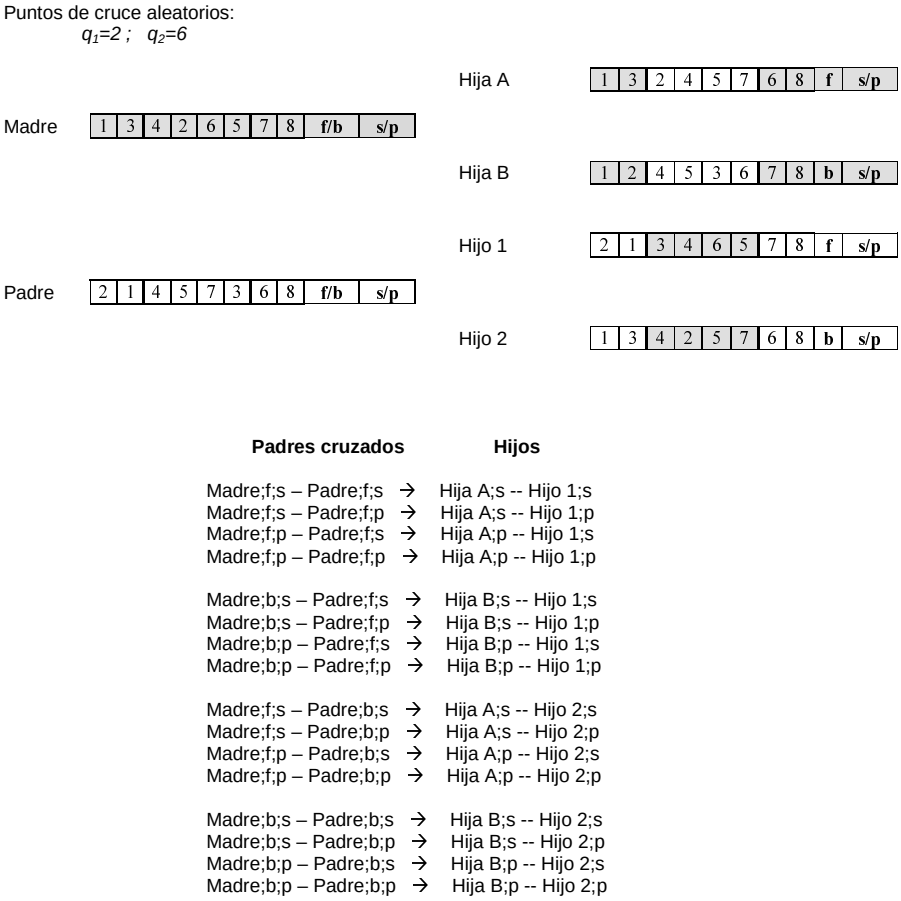

Figura 2: Mecanismo de cruce

2.4. Otros mecanismos

Además de los diseños y procedimientos descritos en los apartados anteriores, propios de los algoritmos genéticos, hemos desarrollado y aplicado a los propios algoritmos otros mecanismos capaces de mejorar la calidad de las soluciones y el comportamiento del algoritmo. En primer lugar desarrollamos una adaptación de un procedimiento de búsqueda local aplicado por distintos autores en diferentes metaheurísticos ([7], [8]) para resolver el problema. La adaptación consiste en hacer uso nuevamente de los bits *f/b* y *s/p* para incrementar todavía más su eficiencia. El segundo mecanismo es una adaptación del procedimiento de reemplazo aleatorio empleado en [9] para la versión multi-modo del problema.

El procedimiento de búsqueda local consiste en realizar una primera pasada sobre el calendario de ejecución representado por la solución justificando dicho calendario a derecha o izquierda, según el bit *f/b* sea *f* ó *b*. Dicha pasada puede

reducir o no la duración del calendario, pudiendo de esta forma mejorar, pero nunca empeorar, la calidad de la solución. Posteriormente puede llevarse a cabo otra pasada sobre el calendario, justificando el mismo en sentido contrario al realizado en la primera fase, lo que puede volver a reducir la duración del mismo.

El procedimiento de reemplazo aleatorio fue diseñado en primer lugar para el algoritmo que resolvía la versión multi-modo del problema, pero posteriormente fue incorporado también al de la versión estándar, comprobando así que también era capaz de mejorar el comportamiento de dicho algoritmo. Este procedimiento se explica en el apartado correspondiente a la versión multi-modo.

2.5. Resultados

Los diseños presentados en los apartados anteriores fueron utilizados para la implementación de un algoritmo genético que comparamos con diversos algoritmos metaheurísticos desarrollados para resolver el problema, comprobando su excelente comportamiento. Un exhaustivo estudio computacional llevado a cabo por Hartmann y Kolisch [12] reveló que nuestro algoritmo genético era uno de los tres mejores algoritmos metaheurísticos desarrollados hasta el momento para resolver el problema.

3. Algoritmos genéticos para la versión multi-modo

En este apartado se describen más brevemente los desarrollos llevados a cabo para resolver la versión multi-modo. La solución al problema consiste ahora en determinar, además de la fecha de comienzo de cada una de las actividades, su modo de ejecución, ya que cada actividad puede presentar diferentes modos de ejecución. Además, en esta versión del problema aparecen recursos no renovables y doblemente restringidos. Los recursos no renovables son aquellos recursos de los que se dispone una cierta cantidad determinada y se consumen a medida que se utilizan. Los doblemente restringidos son aquellos no renovables que además tienen una cantidad máxima de utilización por período. Esta versión más realista del problema complica mucho su resolución, tanto del punto de vista exacto como heurístico.

La codificación diseñada para las soluciones debe incorporar información relativa al modo de ejecución de cada una de las actividades. Así pues, el diseño propuesto consiste en representar la solución por una doble lista, la lista de actividades con el bit f/b y la lista de modos de ejecución de las actividades. La nueva codificación puede observarse en la Figura 3.

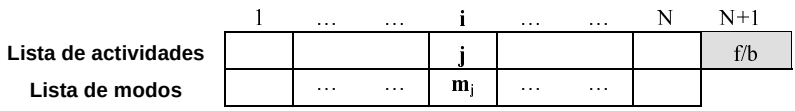

Figura 3: Nueva Codificación, versión multi-modo.

La actividad representada en la posición i de la lista de actividades, será la elegida en i-ésimo lugar para ser secuenciada. Su modo de ejecución estará representado en la misma posición de la lista de modos. Así pues, en cada posición de la lista de modos habrá representado un valor que indique el modo de ejecución de la actividad correspondiente.

Un hecho muy importante a tener en cuenta es que en la versión estándar del problema cualquier lista de actividades que cumpliera las relaciones de precedencia podía transformarse en un calendario de ejecución factible. Es decir, cualquier lista con esa característica se entendía como una solución factible al problema y no se permitían soluciones no factibles. Sin embargo, en la versión multi-modo, una lista de actividades factible, con su correspondiente lista de modos puede no ser una solución factible desde el punto de vista de los recursos no renovables y doblemente restringidos.

Puesto que permitir sólo soluciones factibles complicaría demasiado el algoritmo y no conduciría a un mejor comportamiento del mismo, optamos por seguir el planteamiento presentado por diferentes autores en el que sí se permiten soluciones no factibles (desde el punto de vista de los recursos no renovables), pero éstas son penalizadas en la función objetivo. La forma en la que son penalizadas estas soluciones en distintos trabajos ([10], [11]) consiste en penalizar el valor de la función objetivo de una solución no factible haciendo que sea peor que cualquiera de las soluciones factibles de la población. En concreto, el valor de la función objetivo de una solución no factible únicamente depende del grado de infactibilidad de la misma, pero no recoge para nada la calidad de la solución desde el objetivo de la minimización de la duración del proyecto. Ese era a nuestro entender un punto débil de dicha fórmula, por lo que propusimos una alternativa que sí consideraba tanto el nivel de infactibilidad como la duración del proyecto y que según estudios realizados ofrecía mucho mejores resultados. Los detalles pueden consultarse en [9].

El cruce utilizado es una adaptación del presentado para la versión estándar del problema, siendo los bits f/b de las soluciones cruzadas los que dirigen la operación de cruce. Tras el cruce, cada actividad hereda el modo de ejecución que presentaba en el padre del cual ha sido heredada.

El mecanismo de mutación también es una adaptación del presentado anteriormente. La adaptación consiste en que en una segunda fase, el procedimiento recorre la lista de modos alterando algunos de éstos. Los modos mutan con la misma probabilidad que las actividades, pero de forma independiente.

Para mejorar la eficiencia del algoritmo se diseñó un procedimiento de reemplazo aleatorio, que permitía introducir variabilidad en la población y evitaba que el algoritmo quedase atrapado en óptimos locales. El proceso es aplicado en cada generación con una determinada *probabilidad de reemplazamiento*. Si se lleva a cabo el procedimiento en la población actual, cada solución de la misma es sustituida, con una *probabilidad de intercambio* dada, por una solución generada de forma aleatoria.

En [9] se presenta un estudio computacional en el que se compara el algoritmo genético desarrollado para la versión multi-modo con los mejores metaheurísticos desarrollados hasta el momento, demostrando su eficiencia.

En esta línea de la programación de proyectos, actualmente trabajamos en el desarrollo de algoritmos para resolver otras variantes del problema y al diseño de algoritmos multiobjetivo capaces de resolver problemas que consideran de forma simultánea diferentes objetivos.

Referencias

[1] Brucker P., Drexl A., Möhring R., Neumann K., y Pesch E. (1998). Resource-constrained project scheduling: Notation, classification, models, and methods. *Eur J Opl Res*, **112**, 3-41.

[2] Herroelen W., Demeulemeester E., y De Reyck B. (1998). A classification scheme for project scheduling, in: *Project Scheduling: Recent Models, Algorithms and Applications*, Weglarz J. (ed). Kluwer Academic Publishers, 1-26.

[3] Hartmann S. (1999). *Project Scheduling under Limited Resources*. Springer.

[4] Alcaraz J., y Maroto C. (2001). A robust genetic algorithm for resource allocation in project scheduling. *Ann Op Res*, **102**, 83-109.

[5] Alcaraz J., y Maroto C. (2006). A hybrid genetic algorithm based on intelligent encoding for project scheduling, in: *Perspectives in Modern Project Scheduling: International Series in Operations Research and Management Science* , Jozefowska J., and Weglarz J. (ed). Springer, 249-274.

[6] Boctor F.F. (1996). Resource-constrained project scheduling by simulated annealing. *Intl J Prod Res*, **34**, 2335-2351.

[7] Tormos P., y Lova, A. (2001). A competitive heuristic solution technique for resource-constrained project scheduling. *Ann Opns Res*, **102**, 65-81.

[8] Valls V., Ballestin F., y Quintanilla M.S. (2005). Justification and RCPSP: A technique that pays. *Eur J Opl Res*, **165**, 375-386.

[9] Alcaraz J., Maroto C., y Ruiz R. (2003). Solving the multi-mode resource-constrained project scheduling problems with genetic algorithms. *J Opl Res Soc*, **54**, 614-626.

[10] Jozefowska J. (2001). Simulated annealing for multi-mode resource-constrained project scheduling. *Ann Oper Res*, **102**, 137-155.

[11] Hartmann S. (2001). Project scheduling with multiple modes: a genetic algorithm. *Ann Oper Res*, **102**, 111-135.

[12] Hartmann S., y Kolisch R. (2006). Experimental investigation of heuristics for resource-constrained project scheduling: an update. *Eur J Opl Res*, **174**, 23-37.

About the authors

Javier Alcaraz is Professor in the Department of Statistics, Mathematics and Computer Science at Universidad Miguel Hernández de Elche. His research has been focused in project scheduling, developing metaheuristic algorithms to solve different problems of this field. Other research lines are production scheduling and locational analysis. He has published several articles in some of the most relevant journals in the field of operations research and management science and has participated in a large number of national and international conferences of those fields.

Concepción Maroto is Full Professor in the Department of Applied Statistics and Operations Research and Quality at Universidad Politécnica de Valencia. Teaching: Operations Research in Business Administration and Management. Her main lines of research are Project Management, Modeling and Optimization Techniques for a Sustainable Development and Flexible Production Programming Techniques. She has supervised several doctoral thesis and published a large number of books and articles in those fields, as well as participating in numerous international conferences.