ISSN: 1692-7257 Volumen 1, No. 6 - 2006

Revista Colombiana de
Tecnologías de Avanzada

HEURISTIC FOR SCHEDULING OF PROJECTS WITH RESTRICTION OF RESOURCES

UN HEURÍSTICO PARA PLANEACIÓN DE PROYECTOS CON RESTRICCIÓN DE RECURSOS

Juan C. Rivera, Luis F. Moreno, F. Javier Díaz, Gloria E. Peña

Escuela de Sistemas, Facultad de Minas, Universidad Nacional de Colombia
Cra 80 No. 65 – 223, Bloque M8, Oficina 210, Medellín, Colombia
{jcrivera, lfmoreno, javidiaz, gepena}@unalmed.edu.co

Abstract: The Resource-Constrained Project Scheduling Problem (RCPSP) is a general model for which heuristic algorithms are used, which, although do not guarantee an optimal solution, can give satisfactory results in considerably smaller times to those obtained by means of exact analytical techniques. In this work Taboo Search is used to solve the version of the RCPSP whose main features are: use of constrained renewable resources and impossibility to interrupt the processing of the activities (no pre-emption allowed); the objective is to minimize the makespan of the project. The algorithm is applied to some benchmark problems.

Resumen: El Problema de Planeación de Proyectos con Restricción de Recursos (RCPSP) es un modelo general para el cual se utilizan algoritmos heurísticos, los cuales, aunque no garantizan un óptimo, pueden entregar resultados satisfactorios en tiempos considerablemente menores a los obtenidos mediante técnicas analíticas exactas. En este trabajo se utiliza la Búsqueda Tabú para solucionar la versión del RCPSP cuyas principales características son: utilización de recursos renovables limitados e imposibilidad de interrumpir el procesamiento de las actividades; el objetivo es minimizar la duración del proyecto. El algoritmo se aplica a algunos problemas benchmark.

Keywords: RCPSP, Scheduling, Heuristic, Tabu search, Combinatorial optimization.

1. INTRODUCCIÓN

La secuenciación o programación de tareas es hoy y cada vez mas una actividad de suma importancia en la planeación de proyectos. Entre sus objetivos se encuentra por ejemplo el aumento de la capacidad de las empresas, la disminución de los costos de operación, el aumento de la eficiencia de las empresas, la evaluación de diferentes alternativas tecnológicas y los cambios en los procesos de producción o servicio, entre otros.

Los problemas de planeación de proyectos, los cuales son un caso particular de los de Programación de Actividades, son muy comunes en cualquier tipo de industria o empresa ya que éstos pueden ser aplicados en la programación de producción industrial, proyectos de construcción, servicios a diferentes clientes y a actividades o tareas cotidianas y rutinarias del hombre.

La Programación de Actividades básicamente consiste en la asignación de recursos a actividades en el tiempo, siempre y cuando se cumplan ciertas restricciones de disponibilidad de los recursos y de precedencias entre las actividades. Así, el problema consiste en realizar tal asignación optimizando alguna función objetivo.

A pesar de que la definición dada anteriormente es aparentemente tan sencilla y únicamente cualitativa, da origen a definiciones, formalizaciones y grandes discusiones de tipo matemático cuando se trata de resolver el problema general de Programación de Actividades, el cual se vuelve un problema muy complejo en la medida en que crece su tamaño.

2. DESCRIPCIÓN DEL PROBLEMA

En este documento se considera la versión estándar del *Problema de Programación de Proyectos con Restricción de Recursos*", más conocido como *RCPSP* por sus siglas en inglés (*Resource Constrained Project Scheduling Problem*), donde los recursos son renovables y limitados, no es permitido interrumpir el procesamiento de las actividades y el objetivo es la minimización de la duración del proyecto. Sin embargo el problema sigue siendo bastante general, ya que comprende por ejemplo algunos casos particulares de programación de producción, tales como Flow Shop, Job Shop, Permutation Flow Shop y Open Shop.

La formulación matemática para esta versión del RCPSP se describe en (Mingozzi et al., 1995) como sigue:
Dados un conjunto $X = \{1,...,n\}$ de actividades (o trabajos) y un conjunto de m recursos, donde cada recurso k tiene una disponibilidad total b_k durante cada intervalo de tiempo del período de programación. Cada actividad i tiene un tiempo de procesamiento (duración) d_i y su ejecución requiere una cantidad constante r_{ik} del recurso k durante su duración. Todas las cantidades d_i, r_{ik} y b_k son números enteros no negativos; no es permitido interrumpir el procesamiento de las actividades y se asume que los tiempos de alistamiento están incluidos en los tiempos de procesamiento. Con cada actividad j está asociado un conjunto $\Gamma_j^{-1} \subset X \setminus \{j\}$ de predecesores inmediatos: actividades que deben ser completadas antes de iniciar la ejecución de j. Las restricciones de precedencia pueden estar representadas por un grafo dirigido acíclico $G = (X, H)$ donde $H = \{(i, j) : i \in \Gamma_j^{-1}, j \in X\}$. Se asume, sin perder generalidad, que las actividades están topológicamente ordenadas, es decir, cada predecesor de la actividad j tiene un número de actividad más pequeño que j.

Las actividades 1 y n se usan para representar el inicio y el fin del proyecto: la actividad 1 debe completarse antes de iniciar las actividades $X \setminus \{1\}$ y la actividad n debe empezar después de completar las actividades $X \setminus \{n\}$. Se asume además que $d_1 = d_n = 0$ y $r_{1k} = r_{nk} = 0, \ \forall \ k$.

El costo de una solución factible está dado por el tiempo de terminación del proyecto (*makespan*). El objetivo es encontrar un tiempo de inicio factible para cada actividad que satisfaga las restricciones de precedencias y de recursos, tal que el costo de la solución sea mínimo. A pesar de las variantes descritas en el párrafo anterior, el problema tal como se presenta sigue siendo de mucho interés para los investigadores de los problemas de Programación de Actividades, ya que por ser del tipo *NP-hard* no se conocen algoritmos eficientes para problemas de tamaño moderado. Así, por ejemplo, para muchos problemas de 60 actividades (o más) y 4 recursos no se conocen las soluciones óptimas.

3. ESTADO DEL ARTE

Hasta el presente se han diseñado diversos algoritmos con el fin de resolver de manera exacta el RCPSP descrito en el numeral anterior. Entre estos algoritmos se encuentran los desarrollados por (Gorenstein, 1972; Fisher, 1973; Stinson et al. 1978; Talbot and Patterson, 1978; Patterson, 1984; Christofides et al., 1987; Demeulemeester, 1992; Mingozzi et al., 1995; Simpson and Patterson, 1996; Brucker and Knust, 2000; Dorndorf et al., 2000; Erenguc et al., 2001).

Los algoritmos anteriores tratan de resolver el RCPSP mediante técnicas como la programación lineal entera, programación lineal entera mixta, *branch and bound* (ramificación y acotamiento), relajación lagrangiana y programación dinámica. Sin embargo, la naturaleza combinatoria de estos problemas y su pertenencia a la clase NP-hard (Blazewicz et al., 1983), hacen que sea prácticamente imposible resolverlos en tiempos razonables, aún mediante el uso de los computadores más potentes existentes hoy en día.

La tendencia actual es a desarrollar métodos generales para resolver de modo eficiente clases o categorías de problemas, aunque no necesariamente de manera óptima. Estos métodos son denominados heurísticos. (Osman and Kelly, 1996) explican que los heurísticos son una clase de métodos aproximados que están diseñados para resolver problemas difíciles de optimización combinatoria. Los heurísticos proporcionan un marco general para crear nuevos algoritmos híbridos combinando diferentes conceptos derivados de la inteligencia artificial, evolución biológica y mecanismos estadísticos.

Otros factores que pueden determinar la utilización de los métodos heurísticos son:

- Cuando no existe un método exacto para solucionar el problema
- En caso de que no se necesite una solución óptima
- En caso de que los datos sean poco fiables
- Como paso intermedio en la aplicación de otro algoritmo

En la literatura reciente se destacan para la búsqueda de soluciones del RCPSP, heurísticos como Búsqueda Tabú, Recocido Simulado, Algoritmos Genéticos, Optimización de la Colonia de Hormigas y métodos GRASP.

La Búsqueda Tabú puede ser utilizada para volver más agresivo cualquier procedimiento de búsqueda local. Para tal efecto, toma de la Inteligencia Artificial el concepto de memoria y lo implementa mediante estructuras simples con el objetivo de dirigir la búsqueda teniendo en cuenta la historia de ésta. Es decir, el procedimiento trata de extraer información de lo sucedido hasta la solución actual y actuar en consecuencia.

En este sentido puede decirse que hay un cierto aprendizaje y que la búsqueda es inteligente. El principio de la Búsqueda Tabú podría resumirse tal como se enuncia en (Díaz et al., 1996):

"Es mejor una mala decisión basada en información, que una buena decisión al azar, ya que, en un sistema que emplea memoria, una mala elección basada en una estrategia proporcionará claves útiles para continuar la búsqueda. Una buena elección fruto del azar no proporcionará ninguna información para posteriores acciones".

4. METODOLOGÍA

En este documento se implementó el heurístico Búsqueda Tabú para resolver la versión estándar del RCPSP. Algunas de las ventajas de este heurístico son las siguientes:

- Facilidad de implementación.
- Habilidad para arrojar soluciones relativamente buenas en muchos problemas combinatorios.
- Utilización del concepto de memoria que le evita al algoritmo quedar atrapado en óptimos locales.
- Facilidad de intervención de los usuarios mediante diversos parámetros de entrada.
- Tiempo de procesamiento determinado totalmente por el usuario.
- Combinación de conceptos provenientes de otras heurísticas para la elaboración de algoritmos híbridos.

En la Figura 1 se presenta el pseudo-código del algoritmo desarrollado.

Los 6 principales parámetros utilizados por el algoritmo son los siguientes:

- *Solución Inicial:* Para la obtención de una solución inicial se debe elegir un procedimiento determinístico que permita obtener una buena solución en forma rápida. En este estudio se encontró que la mejor solución inicial para el RCPSP es seleccionar siempre de una lista de actividades elegibles (actividades aún no ejecutadas con todos sus predecesores terminados) las actividades con menor rezago u holgura (rezago se toma como la diferencia entre el tiempo de terminación más lejano y el más temprano). El rezago de la actividad se actualiza en cada periodo para reflejar el cambio en el rezago disponible resultante de haber pospuesto algunas actividades individuales.
- *Forma del vecindario:* La forma de definir el vecindario es de mucha importancia ya que esto permite ponerle ciertos límites a la búsqueda y concentrarse en las mejores regiones. Una buena forma de obtener vecindarios es generar soluciones a partir de simples cambios en la posición de las actividades de la cadena crítica.
- *Tamaño de la Lista Tabú:* La lista tabú es el principal constituyente de la estructura de memoria del algoritmo y guarda los atributos

de las últimas soluciones visitadas para evitar volver a evaluarlas en próximas iteraciones cercanas. El tamaño de dicha lista es un parámetro que está determinado por un número entero el cual indicará a cuántas de las últimas soluciones visitadas se les guardarán sus atributos. Un buen valor para el tamaño de esta lista es de 15 unidades según lo experimentado en este estudio.

```
Paso 1   Lectura de Datos y Parámetros del modelo
         (Datos del problema, Tamaño de la lista tabú,
         Nivel de aspiración, Forma del vecindario,
         Criterio de parada, Lower bound (LBS))
Paso 2
         Inicialización de variables
         [Solución global (SG), Solución actual (SA),
          iteración actual (Iteracion),
          Iteraciones sin mejorar (Iter_asp)]
         Hallar una solución inicial (S0)

         Hacer SG = SA = S0

         Iteracion = 0
Paso 3   Iter_asp = 0
         If ( SG = LBS ) Then
            { Se encontró la solución óptima,
            Ir al paso 6 }

         Generación del vecindario (Entorno cercano a SA)
         [Mejor solución candidata del vecindario (SV)]
         Iteracion = Iteracion + 1
         Iter_asp = Iter_asp + 1
         Seleccionar la mejor solución del vecindario: SV
         If ( makespan (SV) < makespan (SG) ) Then
            { SG = SA = SV,
            Iter_asp = Iteracion }
         Else
               If ( SV Є Lista tabú ) Then
Paso 4            { Buscar otra solución candidata para SV }
               Else
                  { SA = SV,
                  Agregar atributos de SV a la Lista Tabú,
Paso 5            Sacar el elemento mas antiguo de la lista }

         Evaluar el Criterio de Aspiración
         If ( Iter_asp = Nivel de aspiración ) Then
            { Hallar una nueva solución SA en otra región
            del espacio factible, iter_asp = Iteracion }

         Evaluar los Criterios de Parada
Paso 6   If ( SA = LBS ) Then
            { Se encontró la solución óptima, ir la paso 6 }
         Else
            If ( El criterio de parada es satisfecho ) Then
               { Ir al paso 6 }
            Else
               { Ir al paso 3 }
         Entregar informes de resultados
```

Fig. 1. Pseudo-código del algoritmo desarrollado.

- *Cota Inferior (lower bound):* Una cota inferior para el makespan se obtiene usualmente eliminando algunas restricciones del problema y tiene la característica principal de ser siempre inferior al *makespan*. Es un parámetro de mucha importancia principalmente porque si se tiene una cota inferior para el *makespan* y durante el proceso de búsqueda se halla una solución cuyo *makespan* sea igual a dicha cota, esto significaría que esta solución es óptima.

  De las múltiples cotas existentes en la literatura se utiliza la cota LBS (Lower Bound de Stinson) ya que presenta una buena relación entre la calidad (cercanía al óptimo) y la facilidad para hallarla.

- *Criterio de Aspiración:* Es una función que permite en determinado momento alcanzar soluciones que no pertenecen al vecindario. Para lograr esto se debe definir un nivel de aspiración, así cuando dicho nivel se satisface se permite el paso a tal solución. Los niveles de aspiración utilizados en este estudio son el *makespan* para aceptar soluciones de la lista tabú y el número de iteraciones sin mejorar para diversificar la búsqueda.

- *Criterio de Parada:* Este criterio permite detener la búsqueda cuando el usuario lo indique. En este estudio se encontró que detener la búsqueda basados en los niveles de aspiración arroja en promedio mejores resultados.

5. RESULTADOS

Para evaluar la eficiencia del algoritmo se empleó el conjunto de 480 problemas de 32 actividades y 4 recursos, disponible en la librería PSPLIB (Kolisch and Sprecher, 2004), para los cuales se conocen los valores de los *makespan* óptimos que se encuentran disponibles en la librería mencionada anteriormente. Además se utilizó un procesador Pentium 4 de 3.19 GHz y 512 MB de memoria RAM. El programa se desarrolló en el lenguaje Visual C++ 6.0.

La eficiencia del algoritmo está medida como el porcentaje de desviación promedio con respecto al *makespan* óptimo versus el tiempo de procesamiento necesario para hallar dicha desviación. Los resultados obtenidos pueden ser observados gráficamente en la Figura 2.

Como el tiempo de procesamiento depende de las características del procesador, algunos autores han propuesto utilizar el porcentaje de desviación promedio con respecto al *makespan* óptimo correspondiente a un número máximo dado de *schedules* para medir la eficiencia de un algoritmo. Se hallaron los resultados con tres valores para el máximo número de *schedules*, tal como se presenta en la Tabla 1.

Tabla 1. Porcentaje de desviación promedio con respecto al makespan optimo versus máximo número de schedules

Máximo número de schedules	*Porcentaje de desviación promedio*
1.000	1,865
5.000	1,190
50.000	0,631

Utilizando como referencia la Tabla 2, tomada de (Kolisch and Hartmann, 2004), donde se recopilan los datos obtenidos por los investigadores más avanzados a nivel mundial que trabajan en los 480 problemas de 32 actividades y 4 recursos de la librería PSPLIB.

Tabla 2. Resultados recopilados de los diferentes investigadores del mundo

Algorithm	SGS	Reference	max. #schedules 1,000	5,000	50,000
GA, TS – path relinking	both	Kochetov, Stolyar [29]	0.10	0.04	0.00
GA – hybrid, FBI	serial	Valls et al. [75]	0.27	0.06	0.02
GA – forw.-backward	serial	Alcaraz, Maroto [1]	0.33	0.12	–
GA – FBI	serial	Valls et al. [76]	0.34	0.20	0.02
sampling – LFT, FBI	both	Tormos, Lova [73]	0.25	0.13	0.05
TS – activity list	serial	Nonobe, Ibaraki [48]	0.46	0.16	0.05
sampling – LFT, FBI	both	Tormos, Lova [71]	0.30	0.16	0.07
GA – self-adapting	both	Hartmann [23]	0.38	0.22	0.08
GA – activity list	serial	Hartmann [22]	0.54	0.25	0.08
sampling – LFT, FBI	both	Tormos, Lova [72]	0.30	0.17	0.09
sampling – random, FBI	serial	Valls et al. [76]	0.46	0.28	0.11
SA – activity list	serial	Bouleimen, Lecocq [9]	0.38	0.23	–
GA – late join	serial	Coelho, Tavares [11]	0.74	0.33	0.16
sampling – adaptive	both	Schirmer [61]	0.65	0.44	–
TS – schedule scheme	related	Baar et al. [4]	0.86	0.44	–
sampling – adaptive	both	Kolisch, Drexl [34]	0.74	0.52	–
GA – random key	serial	Hartmann [22]	1.03	0.56	0.23
sampling – LFT, $\alpha = 1$	serial	Kolisch [33]	0.83	0.53	0.27
sampling – global	serial	Coelho, Tavares [11]	0.81	0.54	0.28
sampling – random	serial	Kolisch [31]	1.44	1.00	0.51
sampling – LFT, $\alpha = 3$	serial	Kolisch [33]	1.05	0.78	0.56
GA – priority rule	serial	Hartmann [22]	1.38	1.12	0.88
sampling – WCS	parallel	Kolisch [32, 33]	1.40	1.28	–
sampling – LFT, $\alpha = 1$	parallel	Kolisch [33]	1.40	1.29	1.13
sampling – random	parallel	Kolisch [31]	1.77	1.48	1.22
GA – problem space	mod. par.	Leon, Ramamoorthy [41]	2.08	1.59	–

De acuerdo con las Tablas 1 y 2, se encuentra que el algoritmo desarrollado en esta investigación presenta las siguientes características:

- Para 1000 *schedules*, en el puesto número 25, entre 26 posiciones.
- Para 5000 *schedules*, en el puesto número 23, entre 26 posiciones.
- Para 50000 *schedules*, en el puesto número 17, entre 20 posiciones.

Con este resultado se logra mostrar la utilidad práctica del algoritmo, ya que no sólo se demuestra que es aplicable a casos reales empresariales sino que adicionalmente, para este caso específico, mejora los resultados obtenidos por otros métodos desarrollados para este problema particular.

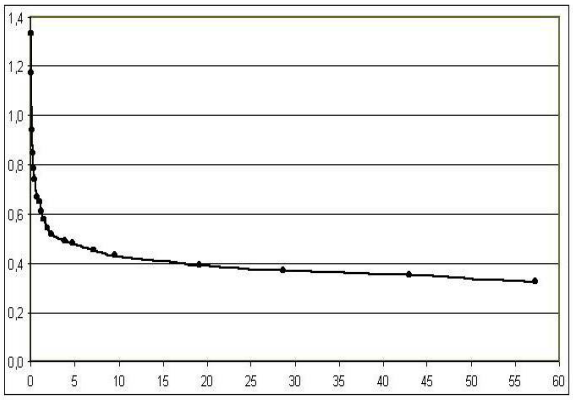

Fig. 2. Gráfica de Desviación porcentual respecto al óptimo vs. Tiempo en segundos para problemas de diferente tamaño

6. CONCLUSIONES

La técnica de la Búsqueda Tabú proporciona excelentes resultados para problemas de tipo combinatorio y existen por explorar muchas otras variaciones y adaptaciones con las cuales se puede obtener cada vez mayor eficiencia. En este estudio se presentó un algoritmo básico de Búsqueda Tabú para hallar soluciones cercanas a la solución óptima de la versión estándar del RCPSP.

El algoritmo presentado es bastante básico ya que aunque contiene los principales elementos de la Búsqueda Tabú como son la lista tabú, la memoria a corto y largo plazo, el criterio de aspiración, las cotas inferiores, etc., no implementa muchas otras estrategias desarrolladas en los últimos años como son los movimientos de influencia, oscilación estratégica, umbrales tabú, entre otras, las cuales implementadas de la manera correcta podrían volver más eficiente el algoritmo.

Otra alternativa que se podría explorar es el desarrollo de algoritmos híbridos que tomen conceptos propios de otras metodologías o incluso de métodos empíricos que no provienen de ninguna metodología específica.

Sin embargo, aún con las estructuras más sencillas se obtuvieron muy buenos resultados para los problemas benchmark utilizados por investigadores

en todo el mundo. Sería de gran utilidad poder comprobar la eficiencia del algoritmo desarrollado con un problema real y así poder analizar todo el potencial de la herramienta.

7. TRABAJO FUTURO

En los diferentes problemas modelo de la Librería PSPLIB se observa que algunos problemas presentan mayor complejidad que otros. Esto puede observarse de diferentes formas en los resultados: por un lado entre mayor sea la diferencia entre el makespan de la solución óptima y el valor de alguna cota inferior, podría pensarse que mayor es la complejidad del problema. Dicha complejidad puede estar relacionada con la cantidad de precedencias entre las actividades y en mayor proporción con la disponibilidad y el consumo de los recursos. Esto lleva a pensar que un análisis profundo de la estructura de cada problema podría dar un indicio de qué tan fácil o difícil podría llegar a ser un problema determinado incluso antes de intentar resolverlo para así controlar el valor de los parámetros de búsqueda en forma automática, dependiendo de la complejidad del problema.

REFERENCIAS

Blazewicz, J.; Lenstra, J. K. and Rinnooy Kan A. H. G. (1983). Scheduling Projects Subject to Resource Constraints: Classification and Complexity. Discrete Applied Mathematics, 5, 11-24.

Brucker, P. and Knust, S. (2000). A linear programming and constraint propagation-based lower bound for the RCPSP. European Journal of Operational Research, 127, págs. 355-362.

Christofides, N.; Alvarez-Valdez, R. and Tamarit, J. M. (1987). *Project Scheduling with Resource Constraints: a Branch and Bound Approach*. Europe Journal of Operations Research, Vol. 29, págs. 262-273.

Demeulemeester, E. (1992). *Optimal Algorithms for various classes of multiple resource constrained project scheduling problems*. Katholieke Universiteit Leuven. Bélgica.

Díaz, A; Glover, F; Ghaziri, H.; González, J.; Laguna, M.; Moscato, P. and Tseng, F. (1996). *Optimización Heurística y Redes Neuronales*. Editorial Paraninfo, págs. 105-142.

Dorndorf, U.; Pesch, E. and Phan-Huy, T. (2000). *A branch and bound algorithm for the resource-constrained project scheduling problem*. Mathematical Methods of Operations Search, 52, págs. 413-439.

Erenguc, S. S.; Ahn, T. and Conway, D. G. (2001). *The resource constrained project scheduling problem with multiple crashable modes: An exact solution method*. Naval Research logistics, 48(2), págs. 107-127.

Fisher, M. (1973). *Optimal Solution of Scheduling Problems Using Lagrange Multipliers. Part I.* Operations Research, Vol. 21, págs. 114-1127.

Gorenstein S. (1972). *An Algorithm for Project Sequencing with Resource Constraints*. Operations Research, Vol. 20, págs. 835-850.

Kolisch, R. and Hartmann, S. (2004). *Experimental Investigation of Heuristics for Resource-Constrained Project Scheduling: An Update*. Technical University of Munich. OR Consulting. Germany.

Kolisch, R. and Sprecher, A. (2004). *Project Scheduling Problem Library – PSPLIB*. [En línea] <http://www.bwl.uni-kiel.de/Prod/psplib/>. Consulta: 13 de febrero de 2004.

Mingozzi, Aristide; Maniezzo, Vittorio; Ricciardelli, Salvatore and Bianco, Lucio. (1995). *An Exact Algoritm for the Resource Constrained Project Scheduling Problem Based on a New Mathematical Formulation*. Department of Mathematics, University of Bologna, Bologna, Italia. Department of Electrical Engineering, University 'Tor vergata', Roma, Italia.

Osman, I. H. and Kelly, J. P. (1996). *Meta-Heuristics: Theory and Applications*. 39 Kluwer Academic Publishers.

Patterson, J. H. (1984). A *comparison of exact approaches for solving the multiple constrained resources, project scheduling problem*. Management Science, Vol. 30, págs. 854-867.

Simpson, W. P. and Patterson, J. H. (1996). *A multiple-tree search procedure for the resource-constrained project scheduling problem*. EJOR, Vol. 89. págs. 525-542.

Stinson, J. P.; Davis, E. W. and Kjumawala, B. M. (1978). *Multiple Resource-Constrained Scheduling Using Branch and Bound*. AIIE Transactions, Vol. 10, págs. 252-259.

Talbot, F. B. and Patterson, J. H. (1978). An Efficient integer programming Algorithm with Network Cuts for Solving Resource-Constrained Scheduling problems. Management Science, Vol. 24, págs. 1163-1174.