Ingeniería y Ciencia
ISSN: 1794-9165
ingciencia@eafit.edu.co
Universidad EAFIT
Colombia

Morillo, Daniel; Moreno, Luis; Díaz, Javier
Metodologías Analíticas y Heurísticas para la Solución del Problema de Programación de Tareas con Recursos Restringidos (RCPSP): una revisión. Parte 2
Ingeniería y Ciencia, vol. 10, núm. 20, julio-diciembre, 2014, pp. 203-227
Universidad EAFIT
Medellín, Colombia

Disponible en: http://www.redalyc.org/articulo.oa?id=83531311012

Ingeniería y Ciencia
ISSN:1794-9165
ISSN-e: 2256-4314
ing. cienc., vol. 10, no. 20, pp. 203–227, julio-diciembre. 2014.
http://www.eafit.edu.co/ingciencia

Metodologías Analíticas y Heurísticas para la Solución del Problema de Programación de Tareas con Recursos Restringidos (RCPSP): una revisión. Parte 2

Daniel Morillo [1], Luis Moreno [2], Javier Díaz [3]

Recepción: 03-05-2013, Aceptación: 10-09-2013
Disponible en línea: 01-07-2014

MSC:90B35

Resumen

En este artículo se exponen y detallan los métodos de solución metaheurísticos más relevantes para el Problema de Programación de Tareas con Recursos Restringidos, RCPSP. Se realiza una revisión crítica del estado del arte, basada en el análisis de los trabajos más significativos publicados en la literatura académica sobre el tema. Inicialmente se presentan diversos métodos metaheurísticos, especialmente aquellos que se han implementado para problemas de secuenciación, destacando sus principales características, así como sus ventajas y desventajas. Además, se presentan los llamados Esquemas Generadores de Secuencias y los índices de complejidad más comúnmente utilizados. Finalmente, se muestra la librería de prueba PSPLIB, usada en la mayoría de trabajos académicos.

[1] Ph.D.(c)., damotor3@posgrado.upv.es, Universitat Politècnica de València, España.
[2] MSc., lfmoreno@unal.edu.co, Universidad Nacional de Colombia, Medellín, Colombia.
[3] Ph.D., javidiaz@unal.edu.co, Universidad Nacional de Colombia, Medellín, Colombia.

Palabras clave: programación de tareas; métodos metaheurísticos; esquemas generadores de secuencias; métodos exactos.

Analytic and Heuristic Methodologies for Solving the Resource Constrained Project Scheduling Problem (RCPSP): a Review. Part 2

Abstract

This paper exposes and details the most relevant methods for the solution of the Resource Constrained Project Scheduling Problem, RCPSP. A critical review of the state of the art, based on the most significant papers published in the academic literature on this topic is carried out. Initially, several metaheuristic methods of solution are shown, especially those that have been implemented for sequencing problems, and their main characteristics, advantages and disadvantages are explained. Furthermore, the so-called sequence generating schemes and the complexity indices most commonly used are presented. Finally, the test library PSPLIB, used in most academic papers related to such problems is considered.

Key words: task scheduling; metaheuristic methods; sequence generating schemes; exact method.

1 Introducción

Dentro de los problemas de secuenciación de actividades, uno de los más estudiados es el de programación de tareas con recursos restringidos (Resource Constrained Project Scheduling Problem: RCPSP). Este es un problema combinatorio NP-hard cuyo propósito es encontrar los tiempos de inicio de un conjunto de actividades que constituyen un proyecto, con la finalidad de minimizar su tiempo de ejecución total, a la vez que deben cumplir dos conjuntos de restricciones: el primero, denominado de relaciones de precedencia y el segundo, de capacidad de recursos renovables.

De acuerdo a [1], el RCPSP puede definirse de la siguiente manera: sea un proyecto constituido por un conjunto $X = (1, \ldots, n)$ de actividades y un conjunto de $S = (1, \ldots, m)$ de recursos, donde cada recurso $k \in S$ tiene una disponibilidad total b_k en cada intervalo de tiempo. Cada actividad $i \in X$ tiene un tiempo de procesamiento constante d_i, cuya ejecución requiere de una cantidad constante r_{ik} del recurso k. Las interrupciones de las actividades no son permitidas y los tiempos de preparación se asumen

dentro de los tiempos de procesamiento e independientes de las actividades. Las actividades ficticias 1 y n representan el inicio y la finalización del proyecto, con duración y consumo de recursos iguales a cero.

2 Enfoques de Solución

Los enfoques que se han usado para la solución del RCPSP son exactos y de aproximación. Los primeros garantizan una solución óptima, siempre que ésta exista; sin embargo, en problemas grandes o muy complejos podría hacerse inviable ya que el tiempo de cómputo crece en forma exponencial con el tamaño del problema. Los segundos, son metodologías heurísticas o metaheurísticas que si bien no garantizan la optimalidad, pueden dar muy buenas soluciones en tiempos razonables. En este artículo se hace hincapié en los métodos metaheurísticos más relevantes reportados en la literatura para la solución del RCPSP, los cuales se enuncian y analizan a continuación.

2.1 Algoritmos Metaheurísticos (Metaheuristic Algorithms)

Los metaheurísticos son procedimientos en los cuales se incorpora una cierta inteligencia que permite tanto la exploración, para la búsqueda en un gran espacio de soluciones factibles, como la explotación, para concentrar la búsqueda en una región reducida del espacio factible donde puede estar la solución óptima. Además, integran mecanismos para escapar de óptimos locales. Estos algoritmos son los más conocidos en la literatura para la solución de múltiples problemas complejos, como los combinatorios, grupo al cual pertenece el RCPSP.

Algunas de las razones de la popularidad y éxito de estos metaheurísticos son su gran versatilidad que permite su implementación para la solución de diversos tipos de problemas, la relativa simplicidad de sus conceptos subyacentes y un amplio espectro de variantes en sus aplicaciones. En [2] se propone la siguiente clasificación en función del tipo de procedimiento utilizado: metaheurísticos de relajación, que eliminan restricciones del problema original; metaheurísticos constructivos, que construyen paso

a paso cada componente de una solución; metaheurísticos de búsqueda, especializados en escapar de óptimos locales; metaheurísticos evolutivos, que usan una búsqueda con varias soluciones simultáneas; metaheurísticos de descomposición, que dividen el problema en sub-problemas cuyas soluciones integradas aportan a la solución del problema original; metaheurísticos de aprendizaje, que usan la información que se va encontrando en el proceso para mejorar la búsqueda.

A continuación se describirán los principales algoritmos metaheurísticos aplicados para la solución del RCPSP.

2.1.1 Temple Simulado (Simulated Annealing) La Metaheurística de Temple Simulado, introducida en [3], se inspira en el proceso de templado del metal, específicamente en el comportamiento molecular de un metal fundido cuando es sometido a un enfriamiento súbito.

El algoritmo empieza desde una solución inicial aleatoria x_0. A partir de ésta, se define una vecindad de soluciones a su alrededor y se evalúa x_0 en la función objetivo $f(x_0)$. Se genera una nueva solución x_1 perteneciente a la vecindad de x_0 y se evalúa la nueva solución $f(x_1)$. Si $f(x_1)$ es mejor que $f(x_0)$, entonces se acepta la nueva solución y se continúa el proceso a partir x_1. De lo contrario, la nueva solución se acepta con una cierta probabilidad, la cual depende de la diferencia de las funciones objetivo, $f(x_0)$ y $f(x_1)$, y de un parámetro T llamado temperatura, la cual debe definirse al inicio del algoritmo y que varía adaptativamente durante todo el proceso. Este procedimiento se repite para n iteraciones o hasta que se cumpla un criterio de parada.

En la expresión (1) se representa el esquema general de la probabilidad de aceptar una solución peor, en el caso de un problema de minimización, donde n es el número de iteraciones y $f(x_n)$ es la función objetivo evaluada en la solución x_n.

$$P(\text{aceptar una solución peor}) = e^{\frac{f(x_{n-1})-f(x_n)}{T}} \tag{1}$$

La expresión (1) muestra que mientras mayor sea la diferencia entre $f(x_{n-1})$ y $f(x_n)$ menor será la probabilidad de aceptación de una nueva solución que no mejore la actual. Para un problema de maximización, el numerador del exponente en la expresión (1) deberá escribirse como

$f(x_n) - f(x_{n-1})$.

La temperatura, por lo general, empieza con un valor grande, lo que permite aceptar con mayor probabilidad soluciones que no mejoran, y con ello explorar el espacio factible en las primeras etapas del algoritmo, evitando estancamientos en óptimos locales. Esta temperatura se reduce de manera gradual a medida que evoluciona el algoritmo, mediante un parámetro r, menor que 1, cuyo valor representa la velocidad de enfriamiento del algoritmo, cambiando su perfil de exploración a uno de explotación. Este procedimiento se repite hasta que se cumpla un criterio de parada. Generalmente, en el temple simulado se define una temperatura mínima, denominada temperatura de congelación, a la cual deberá parar el algoritmo cuando ésta se alcance por primera vez.

2.1.2 Búsqueda Tabú (Tabu Search) Este metaheurístico se describe con detalle en [4]. En esencia, es una búsqueda local que trata de no quedar atrapada en un óptimo local, mediante el uso de información que recauda a medida que el algoritmo transcurre. Por esta razón, la búsqueda tabú es uno de los enfoques más representativos de los algoritmos con memoria.

El algoritmo empieza con una solución inicial aleatoria x_0. Posteriormente, se define un vecindario inicial de soluciones alrededor de x_0 y una lista tabú; luego, mediante reglas de movimiento, se generan nuevas soluciones las cuales se analizan para determinar si se seleccionan. La lista tabú se construye con las soluciones históricamente visitadas, las cuales quedan prohibidas (definidas como tabú), de manera que restringen el vecindario actual garantizando, hasta cierto punto, no volver a seleccionar aquellas soluciones visitadas antes, y así evitar quedar atrapado en un óptimo local. Esta lista tabú es de tamaño limitado, es decir, sólo puede almacenar un número determinado de soluciones; cuando éste se alcanza, se elimina la solución más antigua (que deja de ser tabú) y se adiciona la nueva.

Para aumentar la eficiencia en la etapa de explotación, generalmente, en la lista tabú no se almacenan directamente las soluciones, sino sus características inherentes. Así, cualquier solución que comparta algunas características que estén en la lista tabú no se selecciona. Esta regla solo tiene una excepción: cuando la nueva solución supera la mejor solución encontrada hasta el momento, se le asigna una probabilidad de ser seleccionada, denominada nivel de aspiración.

En [5] se emplean técnicas de inserción para desarrollar un procedimiento de búsqueda tabú para la solución del RCPSP que, de manera iterativa, selecciona una actividad de una solución y la elimina de la programación; luego, la reinserta mediante el uso de reglas de flujo de redes. Tanto la actividad seleccionada como sus predecesores y sucesores se adicionan a la lista tabú. En [6] se diseñan dos esquemas de búsqueda tabú para resolver el RCPSP; el primero genera vecindarios con ayuda de la información de la ruta crítica, y el segundo se basa en la generación de vecinos, como se presenta en [7]. En [8] se propone la llamada Búsqueda Tabú Reactiva (*Reactive Tabu Search*), basada en la representación serial de actividades mediante un Esquema Generador de Secuencias (*Schedule Generator Scheme: SGS*). Las vecindades de soluciones se obtienen con movimientos de intercambio que incluyen desplazamientos de los predecesores y sucesores de las actividades intercambiadas.

2.1.3 Algoritmos Genéticos (Genetic Algorithm) Su trabajo pionero, [9], se inspira en la teoría neo-darwiniana de la evolución, que trasladada al campo de optimización, se puede interpretar así: cada individuo (solución) tiene asociado un valor de su aptitud, representado por su valor en la función objetivo; las diferentes generaciones de individuos evolucionan mediante la selección y operadores genéticos tomando varias soluciones simultáneamente, logrando un proceso de búsqueda en paralelo.

La implementación de estos algoritmos se basa en la representación de las soluciones en un código genético, expresado en lenguaje computacional. En la literatura se conocen dos formas de codificar las soluciones, la representación binaria, propuesta en [9], y la representación en valor real. La diferencia de estos dos métodos, consiste en la manera de definir los denominados operadores genéticos.

En este algoritmo se parte de una población inicial, que generalmente se crea de manera aleatoria. De esta población inicial se seleccionan los padres de la siguiente generación, para lo cual, normalmente se eligen los individuos de menor valor en la función de aptitud (para problemas de minimización) con un alta probabilidad. Sin embargo, también se da cabida a soluciones de mayor valor (soluciones malas), aunque con menor probabilidad, para permitir la exploración en diferentes zonas de la región factible y dar diversidad al algoritmo de búsqueda. El operador genético más impor-

tante es el cruce. La mutación solo cumple un papel secundario aportando, casualmente, cambios aleatorios para permitir la exploración de la búsqueda. Luego, se asignan parejas de padres de manera aleatoria y se procede a usar los operadores genéticos previamente definidos para hallar nuevas soluciones (hijos). Después de la trasferencia de genes y la generación de hijos se usa algún criterio de selección para elaborar la nueva generación. El proceso se repite hasta cumplir con un criterio de parada.

Los algoritmos genéticos son muy utilizados para resolver problemas de programación de proyectos. En [10] se describe detalladamente la implementación de estos algoritmos en este tipo de problemas. Otros trabajos destacados son [11, 12, 13].

2.1.4 Programación Evolutiva (Evolutionary Programming) La programación evolutiva fue propuesta en [14]. En su concepción difiere un poco de los algoritmos genéticos, ya que cada solución no se interpreta como un individuo sino como una especie biológica diferente, y plantea la evolución de manera más global, como cambios en las especies. Por esta razón, el operador genético principal es la mutación, con una probabilidad del 100 %, debido a que el cruce no es posible entre diferentes especies. La ventaja de este enfoque es que mantiene la diversidad de la población [15].

La representación usual es en valor real, donde para cada valor de la solución se define una característica σ_i que representa la desviación estándar de una mutación gaussiana. Este es el parámetro que se utiliza para generar una nueva población. Por este motivo, cada especie i (solución) se compone de una dupla (x_i, σ_i), que se considera como un padre que genera sólo un hijo, a través de la mutación, como se muestra en las expresiones (2) y (3):

$$x_i' = x_i + \sigma_i * N(0,1) \tag{2}$$

$$\sigma_i' = \sigma_i * e^{[\gamma * N(0,1) + \gamma' * N(0,1)]} \tag{3}$$

Donde $N(0,1)$ corresponde a valores aleatorios de una distribución normal estándar con media 0 y varianza 1; n es el número de especies. Los parámetros γ y γ' suelen definirse como $(\sqrt{2\sqrt{n}})^{-1}$ y $(\sqrt{2n})^{-1}$, respectivamente. Para n suficientemente grande puede tomarse $\gamma = 0{,}2$.

Para la selección de los padres se usa una comparación por pares. Este método consiste en definir un número de torneos q, y luego probar cada una de las especies (incluidos los hijos y los padres) de la siguiente manera: cada especie de prueba competirá en q torneos, cuyos oponentes serán seleccionados aleatoriamente; las victorias, entendidas como la solución de mejor función de aptitud, serán almacenadas. Luego se seleccionan las especies que tengan mayor número de victorias; tantas especies como el número que se tenía en la generación anterior.

2.1.5 Greedy Randomized Adaptive Search Procedure: GRASP El metaheurístico GRASP fue propuesto en [16], y es claramente descrito en [17]. Es muy útil en problemas que pueden descomponerse en una secuencia de decisiones que deben tomarse en varias etapas. Este método consta de dos fases fundamentales, una fase constructiva y una de mejora.

En la fase constructiva se generan soluciones factibles de manera iterativa, donde en cada iteración se incorpora un elemento a la solución, cuya selección se basa en el valor de la función de aptitud parcial de cada alternativa disponible. En lugar de seleccionar directamente la mejor alternativa (metodología avariciosa, Greedy) selecciona de manera aleatoria el siguiente elemento a incorporar como parte de la solución, pero asignando una probabilidad en proporción a la función de aptitud. Una vez se elige el elemento a adicionar en la solución, los valores de las alternativas se adaptan a la nueva situación.

En la fase de mejora, una vez encontrada una solución factible, se aplica una búsqueda local para mejorar la solución construida. Como el punto de partida es una solución diferente en cada iteración, se suelen encontrar soluciones muy buenas a través de esta búsqueda local. Estas dos fases se repiten un número definido de veces.

2.1.6 Búsqueda Dispersa (Scatter Search) La búsqueda dispersa fue propuesta en [18] y descrita en [19]. Esta metaheurística se considera un proceso evolutivo donde se construye un conjunto de referencia de soluciones buenas, pero dispersas, es decir, soluciones distanciadas entre sí o significativamente diferentes. Este conjunto evoluciona combinando dos o más soluciones para producir otras mejores, pero manteniendo un significativo grado de dispersión.

Según [20], la búsqueda dispersa consta de 5 elementos, a saber: primero, un método de generación diversificada para generar una colección de soluciones de prueba diversas; segundo, un método de mejora que transforma una solución de prueba en una o más soluciones mejoradas; tercero, un método para construir, mantener y actualizar el conjunto conformado por un número pequeño de las mejores soluciones encontradas; cuarto, un método que, a partir del conjunto de referencia, genera subconjuntos con el fin de combinarlos para crear nuevas soluciones; quinto, un método de combinación de soluciones que transforma un subconjunto dado de éstas en una o más soluciones. Este método de combinación es el equivalente al operador genético de cruce en los algoritmos genéticos.

Este algoritmo se ha implementado para resolver el RCPSP con buenos resultados [21], situándose en la literatura como un método tan competitivo como los algoritmos genéticos para este tipo de problemas.

2.1.7 Optimización de la Colonia de hormigas (Ant Colony Optimization) La colonia de hormigas fue una idea original propuesta en [22] y descrita en [23]. Su algoritmo de optimización es un metaheurístico de poblaciones que intenta reproducir el mecanismo utilizado por las hormigas para localizar el alimento e informar a la colonia donde se encuentra. La ruta seguida por las hormigas tiene la característica de ser la de menor distancia desde su nido hasta el alimento. Este algoritmo se cataloga como un procedimiento de construcción seudo-aleatoria.

Inicialmente, las hormigas se desplazan de manera aleatoria, dejando por cada camino recorrido una cierta cantidad de feromona. Las siguientes hormigas se desplazarán aleatoriamente pero con mayor probabilidad de ir por los caminos que tengan mayor concentración de esta sustancia. Éstas, a su vez, dejan su propia feromona, la cual, con el tiempo, se evapora paulatinamente; es decir, aquellas rutas que no se visiten recurrentemente tienden a desaparecer. De esta manera, algunos caminos se vuelven más atractivos por su alta concentración de feromona.

El principio de este heurístico está basado en que la probabilidad de que una hormiga seleccione una ruta depende del número de hormigas que la hayan seleccionado previamente y de su distancia. Gradualmente, las hormigas preferirán usar las rutas más cortas. El concepto de mínima distancia entre el nido y la fuente de alimento se adaptará a la función de

aptitud, cuyo mínimo valor indicará la ruta a seleccionar. En este algoritmo se construyen soluciones iterativamente.

En la expresión (4) se define la probabilidad de incorporar el elemento j de la solución elaborada hasta el elemento i, de la siguiente manera:

$$P_{i,j} = \begin{cases} \frac{[\tau_{i,j}(t)]^{\alpha} * [\eta_{i,j}]^{\beta}}{\Sigma_{u \notin M_k} [\tau_{iu}(t)]^{\alpha} * [\eta_{iu}]^{\beta}} & \text{si } j \notin M_k \\ 0 & \text{en otro caso} \end{cases} \quad (4)$$

donde:

η_{ij}: parámetro que representa una cierta "visibilidad", relacionada con la función de aptitud. $\tau_{ij}(t)$: rastro de feromona, que depende del tiempo. α: importancia relativa de la feromona. β importancia relativa de la "visibilidad". M_k: memoria asociada a la hormiga k.

En [24] se desarrolló por primera vez una aplicación de la optimización de la colonia de hormigas al RCPSP. Bajo este enfoque, una sola hormiga se define como una aplicación completa para elaborar una solución factible. En cada iteración, la siguiente actividad a seleccionar depende del valor promedio del tiempo de inicio más tardío de cada actividad. Dicho promedio será la feromona que representa el aprendizaje y el efecto de las anteriores hormigas.

2.1.8 Algoritmo de Estimación de la Distribución (Estimation Distribution Algorithm: EDA) El algoritmo de estimación de la distribución, descrito en [25, 26] es un metaheurístico catalogado como poblacional, al igual que los algoritmos genéticos y la programación evolutiva. Sin embargo, a diferencia de éstos, podría decirse que no tiene operadores genéticos, ni de cruce ni de mutación, ni replicación, pero la población va evolucionando de una manera guiada de tal forma que se evite recaer en óptimos locales.

Este algoritmo parte de la estimación de una función de distribución de probabilidad y usa la simulación como mecanismo de evolución, en lugar de manipular directamente los individuos de la población que representan soluciones del problema.

El algoritmo inicia generando una población aleatoria de N individuos. A continuación se realizan iterativamente tres pasos: en el primero, se se-

lecciona un subconjunto de los mejores individuos; en el segundo, se realiza un ajuste de una función de distribución de probabilidad a las características del subconjunto creado en el primer paso; en el tercero, se generan nuevos individuos a través de la función de distribución simulada. El algoritmo se detiene cuando la generación de individuos deja de mejorar de manera significativa.

2.1.9 Algoritmos Híbridos (Hybrid Algorithms) El interés en este tipo de algoritmos se debe a su gran éxito en la solución de problemas NP-Hard, como los combinatorios. La idea fundamental de los algoritmos híbridos es combinar o unir varias metaheurísticas que se complementan entre sí, es decir aprovechar las cualidades de un algoritmo y compensar sus deficiencias con otro que se desempeñe mejor en ese aspecto [27]. Por ejemplo, la gran capacidad de exploración de un algoritmo genético podría unirse a la buena explotación de un algoritmo de búsqueda local.

La forma tradicional de generar un algoritmo híbrido se basa en la utilización de uno de los tres siguientes principios básicos: el primero, remplazar un procedimiento no deseable de un metaheurístico por el de otro particular de mejor desempeño para este procedimiento; el segundo, implementar dos o más metaheurísticos en secuencia, uno después de otro; el tercero, programar dos o más metaheurísticos en paralelo.

Pueden encontrarse numerosos ejemplos de aplicación de algoritmos híbridos para la solución del RCPSP; por ejemplo, es muy común implementar un metaheurístico que mejore la solución encontrada por otro. En [28] se desarrolla un algoritmo genético híbrido, con búsqueda local que incorpora un operador genético específico para el RCPSP y una mejora local que se aplica a todas las secuencias generadas. En [29] se muestra un procedimiento llamado *Forward-Backward Improvement*, FBI, descrito más adelante en este trabajo, que puede mejorar la solución obtenida por cualquier algoritmo metaheurístico que trata de resolver el RCPSP.

2.1.10 Algoritmos Hiperheurísticos (Hyperheuristic Algorithms) Los hiperheurísticos se consideran metaheurísticos de alto nivel, que coordinan y controlan otros metaheurísticos de bajo nivel o heurísticos primitivos, de manera adaptativa según el problema, con el fin de encontrar la mejor manera de combinar varios heurísticos para encontrar la solución. Los hi-

perheurísticos no manipulan soluciones del problema directamente pues en cada momento deciden cómo y qué procedimiento usar para manipular dichas soluciones y mejorar las estrategias de búsqueda [2].

A pesar de tener características similares, los algoritmos hiperheurísticos y los híbridos son tenuemente diferentes a nivel conceptual, en el sentido de la jerarquía que tiene cada uno. En un algoritmo híbrido, los heurísticos asociados son del mismo nivel e intervienen directamente sobre la solución, mientras que en los hiperheurísticos, existe un algoritmo base de alto nivel que controla la manera en que intervendrán los heurísticos de bajo nivel, en el procedimiento de solución.

2.1.11 Algoritmos Adaptativos (Adaptive Algorithms) Los algoritmos adaptativos utilizan información propia del problema particular, que están por resolver, para la estimación de los valores más convenientes para los parámetros. Los algoritmos auto-adaptativos modifican los valores de los parámetros durante su ejecución, con base en la información del problema y en la que haya obtenido el algoritmo hasta el momento.

Los valores de los parámetros definen la estrategia que seguirá el algoritmo en su funcionamiento, regulando si se está explorando en una amplia zona de la región factible o si se está explotando alrededor de una solución local, en un vecindario de soluciones. Un buen algoritmo puede no encontrarse en ninguno de estos extremos; el balance de los valores de estos parámetros determina la calidad de la solución.

Estos algoritmos, más que una categoría diferente, son heurísticos que poseen parámetros con características especiales, tales como la tasa de mutación en un algoritmo genético, la probabilidad de aceptar una solución que no mejora la actual en un enfriamiento simulado, la tasa de evaporación de las feromonas en el caso de optimización de la colonia de hormigas, o algunos de tipo más general como el criterio de parada o el utilizado para elegir el esquema generador de secuencias.

2.2 Heurísticos Basados en Reglas de Prioridad (Priority Rule Based Heuristics)

Dentro de los problemas de secuenciación de actividades (*scheduling*) se encuentran metodologías heurísticas que no tienen como propósito encontrar un óptimo global sino obtener soluciones factibles. Si bien los heurísticos basados en reglas de prioridad no son tan sofisticados como los metaheurísticos, se usan conjuntamente con ellos, especialmente para generar soluciones iniciales.

Algunas metodologías basadas en reglas de prioridad usan la programación obtenida del método de la ruta crítica PERT/CPM, de la cual se obtiene información de varios parámetros de cada actividad, tales como tiempo más próximo de inicio (*early start*), tiempo más tardío de inicio (*late start*), tiempo más próximo de finalización (*early finish*), tiempo más tardío de finalización (*late finish*) y holguras, entre otros.

Los métodos heurísticos basados en reglas de prioridad son procedimientos iterativos compuestos de dos elementos: un esquema generador de secuencias y una regla de prioridad, tal como se describe a continuación.

2.2.1 Esquema Generador se Secuencias (Schedule Generator Scheme: SGS) El Esquema Generador de Secuencias es una metodología que determina la manera de seleccionar cada actividad para elaborar una secuencia factible, asignando tiempo de inicio y de finalización a las actividades. La metodología comprende dos tipos de esquemas, en serie y en paralelo, que generan secuencias factibles extendiendo iterativamente una secuencia parcial, que es aquella donde solo un subconjunto de las actividades tiene asignado un tiempo de inicio.

- Esquema en Serie: Este esquema, descrito en [30], usa un número de etapas igual al número de actividades del proyecto. En cada etapa, se selecciona una actividad según una regla de prioridad y se programa tan pronto como sea posible, siempre y cuando cumpla las restricciones de precedencias y de recursos. El algoritmo termina cuando se hayan programado todas las actividades. En [30] se asegura que el conjunto de soluciones generadas por este algoritmo siempre contendrá la solución óptima.

- Esquema en Paralelo: Este esquema, descrito en [30], consiste en programar tantas actividades como sea posible en cada instante en el tiempo, respetando las restricciones de recursos y de precedencias. Este tiempo se denomina tiempo de decisión o nivel, y está asociado con el tiempo de finalización de alguna actividad en ejecución. La definición de este esquema exige que no exista ningún período tal que una actividad que sea factible, en relación a las restricciones de recursos y precedencias, no sea programada. Aunque estos esquemas en paralelo pueden producir secuencias más compactas, no siempre contienen la solución óptima, pues las actividades programadas en cada nivel pertenecen al primer subconjunto del conjunto potencia, y el óptimo puede no pertenecer a ese subconjunto.

2.2.2 Reglas de Prioridad (Priority Rules) Las reglas de prioridad establecen la actividad que se va a seleccionar durante el proceso de búsqueda heurístico o bien en un esquema generador de secuencias. Estas reglas generan una lista de actividades organizadas según una prioridad, pero respetando las restricciones de precedencia. En [8] se realiza una clasificación de las reglas, según la información que usan en su definición: las primeras, son reglas basadas en la red; las segundas, en la ruta crítica; las terceras en el consumo de recursos; las cuartas, en mezclas de prioridades.

Adicional a estas categorías, las reglas de prioridad suelen clasificarse en otros conjuntos, con base en diferentes características como la frecuencia con la cual se calcula la regla. En este sentido, existen reglas estáticas y dinámicas; las primeras son aquellas que se calculan sólo una vez, antes de empezar la programación; las segundas son las que deben calcularse y actualizarse en cada iteración. En [8] se realiza una revisión de más de 70 reglas de prioridad y se resumen las más importantes y populares que se usan en la literatura.

2.2.3 Método Forward-Backward Improvement (FBI) El *Forward-Backward Improvement*, FBI, también conocido como método de doble justificación, puede utilizarse como complemento de cualquier otro algoritmo metaheurístico, puesto que está diseñado para mejorar una solución factible encontrada previamente mediante otro algoritmo. Se basa en el concepto de holgura, entendida como su tiempo libre, en el cual se puede adelantar

o retrasar una actividad sin alterar el tiempo de finalización del proyecto.

En [29] se explica este método. En esencia, el FBI consiste en hacer al menos una revisión hacia atrás y una hacia adelante. En la primera se listan todas las actividades en orden decreciente de su tiempo de finalización; luego se desarrolla un esquema generador de secuencias en serie, de tal manera que todas las actividades se desplazan a la derecha al tiempo más tardío de finalización (*late finish*). En la revisión hacia adelante, las actividades se listan en orden creciente de su tiempo de inicio y se aplica un esquema generador de secuencias en serie de manera que todas las actividades se desplazan a la izquierda de forma que todas empiecen en el tiempo más temprano (*early start*) que su holgura lo permita. Finalmente se verifica si hubo un mejor tiempo de finalización del proyecto, es decir si hubo una reducción del *makespan*.

Este método suele aplicarse iterativamente a una soluciónencontrada por otra metaheurística, hasta que al realizar una pasada backward y una forward no se obtenga una mejora en el tiempo total de finalización del proyecto.

3 Caracterización del RCPSP e Índices de Complejidad

Es de gran importancia comprender y clasificar las diferentes instancias del RCPSP, con la finalidad de diseñar estrategias eficientes según su complejidad para incorporarlas en los algoritmos de solución, permitiendo de esta manera, introducirles características adaptativas a los mismos.

Como lo explica [31], si pudiera establecerse con claridad, a priori, el nivel de complejidad de una instancia del problema, podrían adaptarse los parámetros del algoritmo para obtener un mejor rendimiento; por ejemplo, cambiar entre estrategias que se inclinen a buscar soluciones por la estructura o por el consumo de recursos. Sin embargo, clasificar y determinar la dificultad de una instancia del RCPSP es una tarea difícil y aún es un tema ampliamente estudiado. A continuación se enuncian los principales indices de complejidad mostrados en la literatura.

3.1 Tamaño del Proyecto (Project Size)

El primer índice que intentó dar una medida de la complejidad de los problemas fue el número de actividades. Por supuesto, este índice no es confiable debido a que no considera la estructura de la red ni la información del uso de recursos. Un mayor número de actividades no implica necesariamente mayor complejidad de una instancia.

3.2 Número mínimo de arcos no redundantes (Minimal number of non-redundant arcs)

En (5), n representa el número total de actividades, incluidas las ficticias. En un grafo del RCPSP, los arcos representan las restricciones de precedencia de sus actividades. Los arcos no redundantes son aquellos que no están implícitos en otros arcos. En este índice se tiene el supuesto de que las tareas están relacionadas en serie. Sólo se basa en el número total de tareas del problema y, por lo tanto, no es muy confiable.

$$A^{min} = (n-1) \tag{5}$$

3.3 Número máximo de arcos no redundantes (Maximal number of non-redundant arcs)

La expresión (6), válida sólo para n>5, permite calcular este índice, el cual considera que las actividades están relacionadas en paralelo. Este índice sólo se basa en el número total de tareas del problema; por lo tanto no es muy confiable.

$$A^{max} = \begin{cases} n-2+\left(\frac{n-2}{2}\right)^2 & \text{si } n \text{ es par} \\ n-2+\left(\frac{n-1}{2}\right) * \left(\frac{n-3}{2}\right) & \text{si } n \text{ es impar} \end{cases} \tag{6}$$

3.4 Complejidad de la Red (Network Complexity)

Este índice de complejidad, expresión (7), es el primero que involucra realmente la estructura de la red del problema, donde A son las precedencias

no redundantes existentes, es decir, todos los arcos en un grafo. De esta manera, este índice define el número medio de las relaciones de precedencias no redundantes por actividad.

$$NC = \frac{A}{n} \tag{7}$$

3.5 Índice de complejidad (Complexity Index)

Este índice de complejidad se calcula luego de aplicar, consecutivamente, en el orden que se mencionan, tres procedimientos de reducción descritos en [32], a saber: reducción en paralelo, reducción en serie y reducción de nodo. De esta manera cualquier grafo no cíclico puede reducirse a un solo arco, luego de un número finito de iteraciones que depende del problema. Entonces, el índice de complejidad puede definirse como el número más pequeño q de una secuencia de reducción de nodos que se aplicó a un RCPSP para reducirlo a un solo arco.

En [32] se desarrolla un algoritmo para encontrar el menor número de reducciones de un grafo para convertirlo en un problema de un solo arco, en un tiempo de cómputo polinomial. A pesar de esto, el cálculo de este índice puede ser complejo y demandar mucho tiempo cuando el proyecto tiene muchas actividades; además, analizando los resultados mostrados en [33], puede observarse que este indicador no explica más del 7 % de la variación del tiempo de procesamiento.

3.6 Factor de Recursos (Resource Factor)

La expresión 8 suministra la proporción media del número de diferentes tipos de recursos para los que cada actividad no ficticia posee una demanda no nula. Es decir, es la proporción media de recursos utilizados por cada actividad. $RF = 1$ implica que todas las actividades utilizan todos los tipos de recursos, mientras que, en un caso hipotético, $RF = 0$ indicaría que ninguna de las actividades consume recursos, y su solución sería equivalente a resolver un problema de ruta crítica.

$$RF = \frac{1}{(n-2)*k} * \sum_{i=2}^{n-1} \sum_{k=1}^{k} \begin{cases} 1 & \text{si } r_{ik} > 0 \\ 0 & \text{en otro caso} \end{cases} \tag{8}$$

K es el número de recursos renovables que se tienen disponibles; n, número total de actividades (incluyendo las ficticias); r_{ik}, cantidad de recurso k consumido por la actividad i.

Este indicador es muy usado en la literatura. Sin embargo, sólo considera las actividades que usan cada tipo de recurso pero no la cantidad consumida; por lo tanto, no utiliza toda la información disponible.

3.7 Intensidad de recursos o fortaleza de recursos (Resource strength)

El indicador que se muestra en la expresión (9) se calcula por cada tipo de recurso y expresa la relación entre el recurso total disponible y su consumo promedio. Valores grandes de RS_k indican que la instancia no tiene demasiado restringido el recurso k, mientras que valores pequeños pueden interpretarse como gran restricción en este tipo de recurso.

$$RS_k = \frac{R_k}{\frac{1}{n-2} * \sum_{i=2}^{n-2} r_{ik}} \tag{9}$$

En [34] se mejora este criterio y se propone la expresión (10) para calcularlo, donde a_k es la cantidad total disponible del recurso renovable tipo k, $r_k^{min} = max_{i=1,\ldots,n-2} r_{ik}$ y r_k^{max} es la demanda máxima del recurso k en la secuencia de la ruta crítica.

$$RS_k = \frac{a_k - r_k^{min}}{r_k{}^m ax - r_k^{min}} \tag{10}$$

Un problema con muy pocos recursos disponibles tendría un $RS_k = 0$, y uno sin restricción en sus recursos tendría un $RS_k = 1$. Dada la definición de r_k^{max}, este criterio usa información asociada a los recursos y a las precedencias dadas por la estructura de la red. En [34] se realizaron múltiples experimentos donde se verificó que, para encontrar la solución del RCPSP, este indicador tenía un gran impacto en los tiempos de procesamiento.

3.8 Restricción de Recursos (Resource Constraint)

Este indicador, definido en la expresión (11), compara el valor promedio de uso del recursos k, con el total de recursos disponibles, donde R_k es la cantidad total disponible del recurso renovable k, y $(\overline{r_k})$ se define en la expresión (12).

$$RC_k = \frac{\overline{r_k}}{R_k} \tag{11}$$

$$\overline{r_k} = \frac{\sum_{i=2}^{n-2} r_{ik}}{\sum_{i=2}^{n-2} \begin{cases} 1 & \text{si } r_{ik} > 0 \\ 0 & \text{en otro caso} \end{cases}} \tag{12}$$

Con el fin de investigar la importancia de cada uno de estos índices y para poder inferir cuáles son los que mayor significancia tienen para caracterizar las instancias del RCPSP, en [35] se realizó un estudio de regresión multivariada de los tres índices más relevantes: NC, RS y RF, considerándolos como las variables independientes y la desviación respecto al óptimo la variable dependiente. Los tres índices fueron significativos; sin embargo, no explicaban más del 40 % de variación.

4 Librería de Prueba PSPLIB

En [36] se propone el algoritmo ProGen, generador de una clase general de problemas de secuenciación y programación de tareas, disponible en: `http://www.om-db.wi.tum.de/psplib/main.html`. Estos problemas se forman de manera aleatoria usando un diseño de experimentos factorial basado en dos conjuntos de parámetros, uno fijo y otro variable.

El conjunto de parámetros fijos hace referencia a los que se mantendrán estáticos durante la generación de todo el conjunto de problemas. Estos parámetros se resumen en [37]. En la Tabla 1 se muestra los parámetros fijos usados para la generación de la librería PSPLIB [34], y en la Tabla 2, un resumen de los parámetros variables: Complejidad de la Red (NC), Factor de Recurso (RF) y Grado de Restricción de los Recursos (RS).

Tabla 1: Valores de los parámetros fijos para la generación de la PSPLIB

	Tareas	**Modos**	**Duración**	**Tipo de recursos**	**Sucesores y predecesores**
Min	30	1	1	4	1
Máx	30	1	10	4	3

Tabla 2: Valores de los parámetros variables para la generación de la PSPLIB

Parámetro	**Valor**			
NC	1,5	1,8	2,1	NA
RF	0,25	0,5	0,75	1,0
RS	0,2	0,5	0,7	1,0

Se consideran parámetros variables aquellos que cambian de conjunto a conjunto. Valores en el consumo de recursos muy restrictivos o muy holgados generan una instancia poco compleja. Valores intermedios que restrinjan parcialmente el uso de recursos por las actividades, generan problemas de alta complejidad.

En la citada librería PSPLIB, específicamente para el RCPSP, existen cuatro conjuntos de problemas: $j30$, $j60$, $j90$ y $j120$ con 30, 60, 90 y 120 actividades respectivamente. Cada conjunto tiene 480 problemas excepto el $j120$ que tiene 600. Para los demás, con cada combinación de valores de la Tabla 2 (48 en total) se generan 10 instancias, abarcando así, todos los niveles de dificultad.

A continuación, en la Tabla 3, tomada de [37], se muestra los resultados de los métodos metaheurísticos reportados por la literatura para la solución del RCPSP usando la PSPLIB. Los resultados de estos algoritmos se resumen mediante la desviación porcentual media sobre la solución optima (%ADOS) para el conjunto $j30$.

Tabla 3: Desviación porcentual respecto al óptimo – conjunto j30 [37]

No.	Heurística	Autores	%ADOS		
			1.000	5.000	50.000
1	GAPS - RK	Mendes et al,	0,06	0,02	0,01
2	GA, TS – PR	Kochetov & Stolyar	0,10	0,04	0,02
3	GA – Descomposition	Debels & Vanoucke	0,12	0,04	0,02
4	Hybrid GA	Alcaraz & Maroto	0,15	0,06	0,01
5	BPGA	Debels & Vanaucke	0,17	0,06	0,02
6	Scatter Search	Debels et al,	0,17	0,11	0,01
7	GA – Forw, backw	Alcacaz et al,	0,25	0,06	0,03
8	Com-RBRS BF/FB	Tormos & Lova	0,25	0,13	0,05
9	GA – Hybrid	Valls et al,	0,27	0,06	0,02
10	RBRS BF	Tomos & Lova	0,30	0,17	0,02
11	GA – AL	Alcaraz & Maroto	0,33	0,12	*
12	GA – FBI	Vallas et al,	0,34	0,20	0,02
13	GA – self-Adapting	Hartmann	0,38	0,22	0,08
14	SA – AL	Bouleimen & Lecocq	0,38	0,23	*
15	TS – activity list	Klein	0,42	0,17	*
16	Sampling – Random	Valls et al,	0,46	0,28	0,11
17	TS – activity list	Nonobe & Ibaraki	0,46	0,16	0,05
18	GA – activity list	Hartmann	0,54	0,25	0,08
19	Sampling – adaptive	Schirmer	0,65	0,44	*
20	GA – late join	Coelho & Tavares	0,74	0,33	0,16
21	Sampling adaptive	Kolisch & Drexl	0,74	0,52	*
21	Sampling adaptive	Kolisch & Drexl	0,74	0,52	*
22	Sampling global	Coelho & Tavares	0,81	0,54	0,28
23	Sampling MLFT	Kolisch	0,83	0,53	0,27
24	TS – Schedule scheme	Baar et al,	0,86	0,44	*
25	GA – random key	Hartmann	1,03	0,56	0,23
26	GA – priority rule	Hartmann	1,38	1,12	0,88
27	Sampling MLFT	Kolisch	1,40	1,29	*
28	Sampling WCS	Kolisch	1,40	1,28	*
29	Sampling random, serie	Kolisch	1,44	1,28	*
30	Sampling random, parallel	Kolisch	1,77	1,48	1,22
31	GA – Problem space	Leon & Ramamoorthy	2,08	1,59	*

5 Conclusiones

Como resultado de la investigación se obtiene una completa revisión del estado del arte respecto a las metodologías iterativas de aproximación para la solución del RCPSP. Se analizan desde las heurísticas más simples hasta las más elaboradas y complejas, como aquellas cuya finalidad es encontrar soluciones factibles para ser usadas en otros algoritmos. Además, se profundiza en los índices de complejidad usados para caracterizar los problemas de RCPSP. Finalmente se propone una clasificación de estas metodologías con el fin de agruparlas de manera lógica y ordenada, debido a que con el surgimiento de nuevos enfoques muchas veces se mezclan conceptos o estos se usan de manera ambigua.

Para solucionar el RCPSP existen diversos métodos metaheurísticos; actualmente los más estudiados son los heurísticos basados en poblaciones como los algoritmos genéticos y los algoritmos híbridos; los dos con muy buenos resultados. Además, la revisión de literatura muestra que la caracterización y desarrollo de un índice de complejidad eficaz, que sea capaz de clasificar los problemas en fáciles y difíciles, es aún un tema muy importante y es fuente de futuras investigaciones.

En este trabajo se evidencia que el estudio en el área de programación de actividades es aún un campo activo tanto en la academia como en la industria. En la primera porque brindar alternativas de solución a problemas NP-Hard contribuye significativamente a incrementar el conocimiento de problemas complejos como el RCPSP. En la segunda porque la planificación y programación de tareas es un pilar fundamental en la gestión de diferentes tipos de empresas.

Referencias

[1] A. Mingozzi, V. Maniezzo, S. Ricciardelli, and L. Bianco, "An Exact Algorithm for the Resource Constrained Project Scheduling Problem Based on a New Mathematical Formulation," *Management Science*, vol. 44, no. 5, pp. 714–729, 1995. 204

[2] J. B. Santana, C. C. Rodríguez, F. C. García López, M. García Torres, B. M. Batista, J. A. Moreno Pérez, and J. M. Moreno Vega, "Metaheurísticas: Una

revisión actualizada." *Universidad de La Laguna, España: Departamento de Estadística, Investigación Operativa y Computación*, 2004. 205, 214

[3] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing." *Science (New York, N.Y.)*, vol. 220, no. 4598, pp. 671–80, May 1983. 206

[4] F. Glover and M. Laguna, "Tabu search," *Boston: Kluwer Academic Publishers*, 1997. 207

[5] C. Artigues, P. Michelon, and S. Reusser, "Insertion techniques for static and dynamic resource-constrained project scheduling," *European Journal of Operational Research*, vol. 149, no. 2, pp. 249–267, Sep. 2003. 208

[6] T. Baar, P. Brucker, and S. Knust, "Tabu search algorithms and lower bounds for the resource-constrained project scheduling problem," *Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization*, pp. 1–18, 1999. 208

[7] P. Brucker, S. Knust, A. Schoo, and O. Thiele, "A branch and bound algorithm for the resource-constrained project scheduling problem," *European Journal of Operational Research*, vol. 107, no. 2, pp. 272–288, Jun. 1998. 208

[8] R. Klein, "Project scheduling with time-varying resource constraints," *International Journal of Production Research*, vol. 38, no. 16, pp. 3937–3952, Nov. 2000. 208, 216

[9] H. J. Holland, "Adaptation in Natural and Artificial Systems," *Unversity of Michigand. Press, Ann Arbor*, 1975. 208

[10] J. Lancaster and M. Ozbayrak, "Evolutionary algorithms applied to project scheduling problems - a survey of the state-of-the-art," *International Journal of Production Research*, vol. 45, no. 2, pp. 425–450, Jan. 2007. 209

[11] J. R. Montoya-Torres, E. Gutierrez-Franco, and C. Pirachicán-Mayorga, "Project scheduling with limited resources using a genetic algorithm," *International Journal of Project Management*, vol. 28, no. 6, pp. 619–628, Aug. 2010. 209

[12] V. Valls, F. Ballestin, and S. Quintanilla, "Justification and RCPSP: A technique that pays," *European Journal of Operational Research*, vol. 165, no. 2, pp. 375–386, Sep. 2005. 209

[13] S. Hartmann, "A self-adapting genetic algorithm for project scheduling under resource constraints," *Naval Research Logistics*, vol. 49, no. 5, pp. 433–448, Aug. 2002. 209

[14] L. J. Fogel, A. J. Owens, and M. J. Walsh, *Artificial Intelligence through Simulated Evolution*. NY: John Wiley & Sons, 1966. 209

[15] X. Yao, Y. Liu, and G. Lin, "Evolutionary Programming Made Faster," *Evolutionary Computation, IEEE Transactions on.*, vol. 3, no. 2, pp. 82–102, 1999. 209

[16] T. A. Feo and M. G. C. Resende, "A Probabilistic Heuristic for a Computationally Difficult Set Covering Problem," *Operations Research Letters*, vol. 8, no. 2, pp. 67–71, 1989. 210

[17] M. G. C. Resende and C. C. Ribeiro, "Greedy randomized adaptive search procedures," *Handbook of Metaheuristics. Kluwer Academic Publishers.*, pp. 219–249, 2003. 210

[18] F. Glover, "A Template for Scatter Search and Path Relinking," *Artificial Evolutions, Lecture Notes in Computer Science*, vol. 1363, no. 1, pp. 15–53, 1998. 210

[19] M. Laguna and R. Marti, *Scatter Search: Methodology and Implementations in C*. Publishers Kluwer Academic, Springer US, Dec. 2002. 210

[20] F. G. Ballestin, "Nuevos Métodos de Resolución del Problema de Secuenciación de Proyectos con Recursos Limitados," Ph.D. dissertation, Universidat de Valencia, 2002. 211

[21] D. Debels, B. De Reyck, R. Leus, and M. Vanhoucke, "A hybrid scatter search/electromagnetism meta-heuristic for project scheduling," *European Journal of Operational Research*, vol. 169, no. 2, pp. 638–653, Mar. 2006. 211

[22] M. Dorigo, "Optimization, Learning and Natural Algorithms," Ph.D. dissertation, Politecnico di Milano, Italy, in Italian, 1992. 211

[23] C. Blum, "Ant colony optimization: Introduction and recent trends," *Physics of Life Reviews*, vol. 2, no. 4, pp. 353–373, Dec. 2005. 211

[24] D. Merkle, M. Middendorf, and H. Schmeck, "Ant colony optimization for resource-constrained project scheduling," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 4, pp. 333–346, Aug. 2002. 212

[25] P. Larrañaga and J. A. Lozano, "Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation," *Genetic Algorithms and Evolutionary Computation*, vol. 2, p. 416, 2002. 212

[26] P. Larrañaga, J. A. Lozano, and H. Mühlenbein, "Algoritmos de Estimación de Distribuciones en Problemas de Optimización Combinatoria," *Inteligencia Artificial. Revista Iberoamericana de Inteligencia Artificial*, vol. 7, no. 19, pp. 149–168, 2003. 212

[27] E. G. Talbi, "A Taxonomy of Hybrid Metaheuristics," *Journal of Heuristics, Kluwer Academic Publishers.*, vol. 8, no. 1, pp. 541–564, 2002. 213

[28] V. Valls, F. Ballestin, and S. Quintanilla, "A hybrid genetic algorithm for the resource-constrained project scheduling problem," *European Journal of Operational Research*, vol. 185, no. 2, pp. 495–508, Mar. 2008. 213

[29] P. Tormos and A. Lova, "A Competitive Heuristic Solution Technique for Resource-Constrained Project Scheduling," *Annals of Operations Research*, vol. 102, no. 1-4, pp. 65–81, 2001. 213, 217

[30] R. Kolisch, "Serial and parallel resource constrained project scheduling methods revisted: Theory and computation," *European Journal of Operational Research*, vol. 90, no. 1, pp. 320–333, Apr. 1995. 215, 216

[31] L. F. Moreno, "Desarrollo de una herramienta analítica - heurística para resolver el problema de la programación de tareas (scheduling)," *Informe de año Sabático, Universidad Nacional de Colombia - Sede Medellín.*, 2005. 217

[32] W. W. Bein, J. Kamburowski, and M. F. M. Stallmann, "Optimal reduction of two-terminal directed acyclic graphs," *SIAM Journal on Computing*, vol. 21, no. 6, pp. 1112–1129, 1992. 219

[33] B. De Reyck and W. Herroelen, "On the use of the complexity index as a measure of complexity in activity networks," *European Journal of Operational Research*, vol. 91, no. 2, pp. 347–366, Jun. 1996. 219

[34] R. Kolisch, A. Sprecher, and A. Drexl, "Characterization and generation of a general class of resource-constrained project scheduling problems: Easy and hard instances," *Research report No. 301, Institut für Betriebswirtschaftslehre, Christian-Albrechts-Univerität zu Kiel.Germany.*, 1992. 220, 221

[35] S. Hartmann and R. Kolisch, "Experimental evaluation of state-of-the-art heuristics for the resource-constrained project scheduling problem," *European Journal of Operational Research*, vol. 127, no. 2, pp. 394–407, Dec. 2000. 221

[36] R. Kolisch and A. Sprecher, "PSPLIB - A project scheduling library," *European Journal of Operational Research*, vol. 96, pp. 205–216, 1996. 221

[37] M. Cervantes, "Nuevos Métodos Meta Heurísticos para la Asignación Eficiente, Optimizada y Robusta de Recursos Limitados," Tesis doctoral, Universidad Politécnica de Valencia, 2009. 221, 222, 223