Algoritmo heurístico híbrido con múltiples vecindarios y recocido simulado para resolver el RCPSP

Hybrid Variable Neighborhood and Simulated Annealing Heuristic Algorithm to Solve RCPSP

Juan Carlos Rivera[1*], *Ana Josefina Celín*[2]

[1]Departamento de Ciencias Básicas, Universidad Eafit, Carrera 49 N.° 7 sur-50, Bloque 38, oficina 424, Medellín, Colombia

[2]Facultad de Ingeniería, Universidad de Antioquia, Calle 67 N.° 53-108, Bloque 21, oficina 404, Medellín, Colombia

(Recibido el 2 de febrero de 2009. Aceptado el 13 de septiembre de 2010)

Resumen

En este artículo se presenta un algoritmo heurístico híbrido para resolver el Problema de Programación de Proyectos con Recursos Limitados (RCPSP). El algoritmo diseñado combina elementos de Recocido Simulado y Búsqueda en Múltiples Vecindarios. Adicionalmente, utiliza el método denominado Justificación, el cual es un método diseñado específicamente para el RCPSP. Para evaluar el desempeño del algoritmo se realizó un análisis estadístico para el ajuste de parámetros. Los resultados se comparan con los reportados en la literatura científica.

----- *Palabras clave:* recocido simulado, algoritmos heurísticos, búsqueda en vecindarios variables (VNS), justificación, RCPSP, programación de producción

Abstract

This paper presents a hybrid heuristic algorithm for solving the Resource Constrained Project Scheduling Problem (RCPSP). The algorithm designed combines elements of Simulated Annealing and Variable Neighborhood Search. Additionally, it uses the method called Justification, which is a method designed specifically for the RCPSP. To evaluate the performance of the algorithm, a statistical analysis for tuning the parameters has done. The results were compared with those reported in the scientific literature.

----- *Keywords:* Simulated Annealing, Heuristic Algorithms, Variable Neighborhoods Search (VNS), Justification, RCPSP, Scheduling

* Autor de correspondencia: teléfono: + 57 + 4 + 261 95 00 ext. 9806, fax: + 57 + 4 + 266 42 84, correo electrónico: jrivera6@eafit.edu.co. (J. Rivera)

Introducción

Los problemas de planeación de proyectos son muy comunes en cualquier tipo de industria u organización, ya que pueden ser aplicados en la programación de producción industrial, proyectos de construcción, prestación de servicios, actividades cotidianas y rutinarias, entre otras.

La programación de proyectos con recursos limitados es un ambiente en el cual se deben procesar un conjunto de actividades sujetas a restricciones de precedencias y recursos, siendo estos últimos compartidos por varias 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 tareas.

Definición del problema

El problema de programación de proyectos con recursos limitados (RCPSP) puede ser descrito matemáticamente de la siguiente manera (basado en las definiciones de [1-3]):

Se tiene un conjunto $J = \{1, \ldots, n\}$ de actividades a ser procesadas. Asociado a cada actividad $j \in J$, existe una duración d_j. Además, las actividades están relacionadas mediante restricciones de precedencia, siendo $P_j \in J\{j\}$ el conjunto de todas las actividades predecesoras inmediatas de la actividad j. Las restricciones de precedencia pueden estar representadas por un grafo dirigido acíclico $G = (J, H)$ donde $H = \{(i, j)/i \in P_j, j \in J\}$ Adicionalmente, existe un conjunto $k = \{1,\ldots,m\}$ de tipos de recursos renovables, donde cada tipo de recurso $k \in K$ tiene una capacidad total R_k durante cada intervalo de tiempo del período de programación. Cada actividad j requiere una cantidad constante de r_{jk} unidades del recurso de tipo k durante todo el intervalo de su duración. Se asume, sin pérdida de generalidad, que $r_{jk} \leq R_k$, lo cual garantiza la existencia de soluciones factibles.

Todas las cantidades d_j, r_{jk} y R_k son números enteros no negativos para todo $j \in J$ y todo $k \in K$.

No se permite interrumpir el procesamiento de las actividades y se asume que los tiempos de alistamiento están incluidos en los tiempos de procesamiento. Una vez que un recurso es ocupado por una actividad, no será liberado hasta que la actividad sea realizada.

Las actividades 1 y n son actividades ficticias, usadas con el objetivo de representar el inicio y la finalización de los proyectos, respectivamente. Se asume además que $d_1 = d_n = 0$ y $r_{1k} = r_{nk} = 0$, para todo $k \in K$.

El objetivo es encontrar un programa que se satisfaga las restricciones de precedencia y de recursos, y que minimice la duración del proyecto (makespan).

De acuerdo con [4], el RCPSP es el problema más importante en Programación de Proyectos. Además, el RCPSP es una generalización de otros problemas de programación de producción como Flow Shop, Job Shop y Open Shop ([5, 6]).

Aunque este problema no es el más general, ya que usa tiempos de procesamiento determinísticos para las actividades, los recursos utilizados son renovables (no considera recursos no renovables que involucran el manejo de inventarios y costos), no se permite interrumpir el procesamiento de las actividades, entre otras características, resolverlo eficientemente es aún de gran interés para la comunidad científica ya que, por su pertenencia a la clase de problemas NP-Hard [5], es un problema difícil de resolver para el cual no se conocen algoritmos que lo resuelvan exactamente de manera eficiente. Una formulación matemática para el RCPSP es presentada en [1]:

$$\text{Min } z_p = \sum_{t=es_n}^{ls_n} t \times \varepsilon_{nt} \qquad (1)$$

Sujeto a:

$$\sum_{t=es_j}^{lsj} \varepsilon_{jt} = 1; \; j \in j \qquad (2)$$

$$\sum_{t=es_i}^{ls_i} t \times \varepsilon_{it} - \sum_{t=es_j}^{ls_j} t \times \varepsilon_{jt} \geq d_j; (i,j) \in H \quad (3)$$

$$\sum_{j \in J} \sum_{\tau=\sigma(t,j)}^{t} r_{jk} \times \varepsilon_{j\tau} \leq R_k; t = 0,\ldots,T_{max};\ k = 1,\ldots,m \quad (4)$$

$$\varepsilon_{jt} \in \{0,1\};\ j \in J;\ t = es_j,\ldots,ls_j \quad (5)$$

Donde:

ε_{jt}: variable de decisión binaria, 1 si y sólo si la actividad j empieza al principio del período t.

ls_j: late start de la actividad j (tiempo de inicio más tardío).

es_j: early start de la actividad j (tiempo de inicio más temprano).

t: Representa cada uno de los periodos del horizonte de planeación del proyecto.

$\sigma(t, j) = \max(0, t - d_j + 1)$.

T_{max}: Denota un upper bound o límite superior para el tiempo de terminación del proyecto. Puede ser fácilmente calculada como $T_{max} = \sum_{j \in J} d_j$.

La ecuación (1) representa la función objetivo de cada modelo respectivamente, makespan o duración del proyecto. Las ecuaciones (2) representan el tiempo de inicio de cada actividad en el modelo de programación binaria, el cual debe ser único dada la característica de no interrupción de las actividades, es decir, las que obligan a que una actividad, una vez iniciada, debe continuarse hasta su terminación. Las ecuaciones (3) representan las restricciones de precedencia en cada modelo respectivamente, una actividad sólo puede iniciar una vez terminadas todas las actividades predecesoras. Las ecuaciones (4) representan las restricciones de recursos de cada modelo respectivamente; en cualquier tiempo la cantidad de recursos utilizados por todas las actividades en ejecución no debe superar la disponibilidad de cada tipo de recurso correspondiente. Las ecuaciones (5) indican que las variables de decisión de la formulación binaria únicamente pueden tomar los valores 0 ó 1.

Metodología

Para solucionar el RCPSP se han utilizado diversos enfoques, los cuales se pueden clasificar como métodos exactos y métodos heurísticos.

Métodos exactos

Entre los métodos exactos se encuentran los desarrollados por [1,7-21]. La principal característica de los anteriores algoritmos es que obtienen una solución óptima al problema, cuando ésta existe.

Sin embargo, la naturaleza combinatoria de estos problemas y su pertenencia a la clase de problemas *NP-Hard*, hace que sean prácticamente imposibles de resolver en tiempos razonables, aún mediante el uso de los computadores más potentes existentes hoy en día. La experiencia de otros autores indica que sólo es posible obtener la solución mediante métodos exactos para instancias del problema con menos de 60 actividades [3]. Sin embargo, un proyecto de 60 actividades se puede considerar pequeño si se compara con los casos reales; por lo tanto, toda discusión acerca de la solución del RCPSP se centra en algoritmos aproximados que cada autor, desde su enfoque, plantea como eficientes.

Métodos metaheurísticos

En contraposición a los métodos exactos, los metaheurísticos no garantizan la obtención de soluciones óptimas. Sin embargo, suelen obtener soluciones satisfactorias en tiempos de procesamiento generalmente polinomiales.

Los métodos metaheurísticos más usados son Búsqueda Tabú, Recocido Simulado, Búsqueda en Vecindarios Variables, Algoritmos Genéticos y Optimización de Colonias de Hormigas. A continuación se describirá el Recocido Simulado y la Búsqueda en Vecindarios Variables, los

cuales tienen relación con el método híbrido propuesto es este estudio.

Recocido simulado

El Recocido Simulado (Simulated Annealing, en inglés), descrito en [22-24], es el principal representante de los métodos con criterios de aceptación probabilística. Fue propuesto por primera vez en Metropolis et al. [25], con el fin de simular la evolución de un sólido en un tratamiento térmico basándose en técnicas de Monte Carlo. Posteriormente fue usada en optimización combinatoria en [24].

Una manera de encontrar los estados de energía de sistemas complejos, tales como sólidos, es utilizando la técnica del recocido, en la que el sólido se calienta a una temperatura en la que sus granos deformados recristalizan para producir nuevos granos, y luego se enfría lentamente de forma controlada y de esta manera, cada vez que se baja la temperatura, las partículas se reacomodan en estados de más baja energía hasta que se obtiene un sólido con sus partículas acomodadas conforme una estructura cristalina lo más regular posible (estado fundamental). En la fase de enfriamiento del proceso de recocido, para cada valor de la temperatura debe permitirse que el sólido alcance el equilibrio térmico.

Las leyes de la termodinámica establecen que, a una temperatura T, la probabilidad de un aumento de energía de magnitud ∂E está dada por la expresión (6) (probabilidad de Boltzmann):

$$P(\partial E) = e^{\frac{-\partial E}{kT}} \quad (6)$$

Donde k es la constante de Boltzmann [22].

Así pues, a modo de analogía, en el algoritmo Recocido Simulado, los estados del sistema corresponden a las soluciones del problema, la energía de los estados a los criterios de evaluación de la calidad de la solución, el estado fundamental a la solución óptima del problema, y la temperatura corresponde a una variable de control.

El algoritmo de Metropolis consiste en generar un vecino a la solución actual, calcularle su energía y aceptar ese vecino si tiene menor energía o aceptarlo con mayor energía con cierta probabilidad que depende de la temperatura (T). La probabilidad de aceptación se expresa como sigue:≥

$$P_T\{acepta\ j\} = \begin{cases} e^{\left(\frac{1}{\frac{f(i)-f(j)}{T}}\right)} & si\ f(j) < f(i) \\ & si\ f(j) \geq f(i) \end{cases} \quad (7)$$

De la expresión anterior se puede notar que para valores altos de T, hay mayor probabilidad de aceptar soluciones que no mejoren la función objetivo (comportamiento similar a una búsqueda aleatoria, aceptando fácilmente cualquier solución), mientras que para valores pequeños de T, la probabilidad de aceptar soluciones malas es baja (comportamiento similar a una búsqueda local, aceptando únicamente soluciones que mejoran la solución actual). Por lo tanto es importante que la temperatura baje a una velocidad apropiada para que le permita al método buscar por diversas zonas e intensificar la búsqueda en aquellas zonas prometedoras. Así, el éxito del Recocido Simulado se basa en la escogencia de una buena temperatura inicial (T_O) y una adecuada velocidad de enfriamiento (r).

El Recocido Simulado ha sido utilizado para resolver el RCPSP por [26-31].

Búsqueda en vecindarios variables

La Búsqueda por Entornos o Vecindarios Variables (Variable Neighborhood Search, VNS, en inglés), descrita en [22, 32], es una metaheurística que considera distintas estructuras de vecindarios y las cambia sistemáticamente para escapar de los mínimos locales. El VNS básico, a partir de una solución inicial, ejecuta una Búsqueda Local cuyo procedimiento consiste en reemplazar la solución actual si ha habido una mejora o modificar la estructura del vecindario en caso contrario.{

VNS se basa en tres hechos simples [32]: Un óptimo local con respecto a una estructura de

vecindad no necesariamente lo es con respecto a otra; un óptimo global es un óptimo local con respecto a todas las posibles estructuras de vecindad; y en muchos problemas, los óptimos locales con respecto a una o varias estructuras de vecindad están relativamente cerca.

La Búsqueda en Vecindarios Variables ha sido utilizada para resolver el RCPSP por [33].

Justificación

La justificación es una técnica sencilla y rápida que al aplicarla sobre una solución del RCPSP produce otra de igual o menor duración. [34] introdujo los conceptos de justificación a la izquierda y a la derecha, aunque el objetivo era extender los conceptos de holgura y ruta crítica del caso de recursos no limitados al caso con limitación de recursos. La siguiente descripción del procedimiento está basada en la presentada en [3]:

Dado un programa S, definido por los tiempos de inicio S_j de cada actividad j, justificar una actividad $j \neq n$ a la derecha consiste en obtener un programa S' tal que $s'_i = s_i$, para $i \neq j$ se busca un nuevo tiempo de inicio ($s'_j \geq s_j$) de tal forma que s'_j sea tan grande como sea posible sin aumentar el makespan y sin violar las restricciones de precedencia. La justificación a la derecha de las actividades j en orden decreciente de su tiempo de terminación ($f_j = s_j + d_j$) genera un programa activo a la derecha S^R, que es llamado la justificación a la derecha de S. S^R no es único, ya que depende de la(s) regla(s) de desempate usada(s). Similarmente, el procedimiento anterior se puede realizar en sentido contrario para justificar la solución a la izquierda, aunque esta alternativa no fue considerada en esta investigación.

El procedimiento asegura que la nueva solución obtenida tiene un makespan menor o, en el peor de los casos, igual al de la solución antes de la justificación.

En [3], se realiza un estudio en el que se incorpora esta técnica en 22 algoritmos diferentes y encontraron que siempre hubo un notable mejoramiento en la calidad de la solución. En [35] se presenta un algoritmo genético que incorpora ideas similares y obtiene en muchos casos, las mejores soluciones encontradas hasta ahora.

Algoritmo propuesto

El algoritmo propuesto, es un procedimiento híbrido que combina conceptos de los metaheurísticos de Recocido Simulado, Búsqueda en Vecindarios Variables y la heurística Justificación.

Cada solución factible del problema es representada por una lista de actividades en donde se indica el orden en el que serán programadas cada una de las actividades del proyecto, y cada actividad será programada de tal forma que su tiempo de inicio sea lo más temprano posible, respetando las restricciones de precedencia y de recursos.

A continuación se describen los principales componentes del algoritmo:

Solución inicial

Como solución inicial se utiliza el método presentado en [36], el cual consiste en programar las actividades en el orden de menor rezago u holgura. El rezago de las actividades es recalculado cada vez que se programa una actividad para considerar los recursos requeridos por la actividad programada. Una comparación de la calidad de esta solución con otros métodos constructivos permitieron concluir que la solución de [36] es, en promedio, de mayor calidad, razón por la que se utiliza en esta investigación.

Vecindarios de búsqueda

El procedimiento usa tres vecindarios de búsqueda, los cuales se describen a continuación:

Intercambio: Se seleccionan dos actividades, en las posiciones i e j. La nueva solución será generada al intercambiar las posiciones de dichas actividades; la actividad en la posición

i se ubicará en la posición de la actividad *j*, y viceversa. Para garantizar la factibilidad de la solución resultante se debe verificar que no existan relaciones de precedencia entre las actividades en las posiciones *i* e *j*, y entre éstas y las actividades que se encuentran en posiciones intermedias.

Inserción hacia delante: Se selecciona una actividad en la posición *i*, e una posición *j*, $i \geq j$. La nueva solución será generada al colocar la actividad de la posición *i* en la posición *j*, y las actividades en las posiciones intermedias se mueven una posición hacia la derecha. Para garantizar la factibilidad de la solución resultante se debe verificar la no existencia de relaciones de precedencia entre la actividad originalmente ubicada en la posición *i*, e las actividades ubicadas entre *i* e *j*. Una ventaja de este vecindario, con relación al anterior, es que requiere evaluar menos relaciones de precedencia para garantizar la factibilidad de las soluciones generadas.

Inserción hacia atrás: Es similar al vecindario anterior, pero se debe seleccionar $i \leq j$. La existencia de los dos últimos vecindarios se justifica debido a que los dos vecindarios juntos serían demasiado grandes.

Al seleccionar en cada vecindario las actividades en la posición denominada *i*, se tienen en cuenta solamente las actividades que pertenecen a la ruta crítica (teniendo en cuenta el uso de recursos).

Adaptación

Debido a que no se tiene ningún criterio para seleccionar el orden en que se deben usar los vecindarios y que, por experiencia, dichos vecindarios no son igualmente exitosos en todas las instancias del problema, se optó por una estrategia adaptativa en la que, en cada iteración, se selecciona aleatoriamente el vecindario a ser utilizado de acuerdo a la probabilidad de selección de cada uno.

Inicialmente, cada vecindario tiene la misma probabilidad de ser seleccionado. En la medida en que cada vecindario es utilizado, si éste es exitoso la probabilidad de utilizarlo en las iteraciones futuras aumenta. Por el contrario, si el vecindario no es exitoso, dicha probabilidad disminuye.

La finalidad de esta estrategia es permitir que el método seleccione las probabilidades más adecuadas para cada vecindario de acuerdo a la instancia a resolver.

Selección de soluciones

Con el fin de permitir que la búsqueda escape de regiones de óptimos locales, es permitido aceptar soluciones de no mejora.

Para realizar dicha aceptación, se toma del procedimiento de Recocido Simulado, el concepto de aceptación probabilística. Sin embargo, se introducen grandes cambios con respecto a la mecánica original del Recocido Simulado.

Para cada tipo de vecindario se generan todas las soluciones pertenecientes a éste y, si la mejor solución del vecindario es mejor que la mejor solución conocida, se acepta. En caso contrario se selecciona aleatoriamente una solución del vecindario. La probabilidad de selección de cada solución se calcula de acuerdo con las siguientes ecuaciones:

$$p_i = \frac{p'i}{\sum_{h \in H} p'_h} \tag{8}$$

$$p'_i = \alpha\, x\, e^{Bx \frac{-\partial M}{T}} \tag{9}$$

donde

∂M es la diferencia entre el Makespan de la mejor solución conocida y el de la solución *i*,

T es un parámetro de temperatura similar al usado en el Recocido Simulado.

α y β son dos parámetros de usuario. Estos parámetros son detallados más adelante.

Se debe tener en cuenta que, análogamente al algoritmo de Recocido Simulado, valores grandes de *T*, generan probabilidades de selección

equilibradas; mientras que valores pequeños, generan probabilidades de selección mayores para las soluciones de mayor calidad.

El parámetro *T* disminuirá su valor a medida que avanza el proceso de búsqueda. En cada iteración el valor de *T* será actualizado como sigue:

$$T = T\,x\,\rho \qquad (10)$$

Donde $\rho \in [0,1]$ es análogo a la tasa de enfriamiento de la temperatura en el Recocido Simulado.

Justificación

Una vez obtenida una solución de un vecindario, se aplica el procedimiento Justificación, el cual, cómo se detalla en la sección de resultados, mejora notablemente el desempeño del algoritmo sin aumentar el costo computacional.

Cotas inferiores

Cuando se presentan problemas de minimización es común el uso de cotas inferiores, las cuales son una aproximación a la solución óptima, la cual tiene como característica principal que se puede demostrar que siempre es menor o igual que la solución óptima.

Las cotas inferiores son importantes por dos razones: 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 implica que la solución hallada es óptima y, adicionalmente, puede usarse como aproximación al makespan óptimo para estimar la calidad de una solución.

Algunas de las cotas inferiores diseñadas para el RCPSP se pueden encontrar en [1, 18, 37, 38]. En esta investigación se utilizará la LBS [1] ya que es muy fácil de implementar y sus valores son de buena calidad. Para efectos de comparación y medición de la calidad de los resultados se utilizará la LB0 ya que es la más conocida por los investigadores, es fácil de calcular y se conocen las desviaciones de muchos algoritmos con respecto a dicha cota inferior.

Criterio de parada

Debido a que el procedimiento no se detiene cuando encuentra soluciones óptimas locales, requiere un criterio de parada externo para detener la búsqueda.

En esta investigación, se utilizó el número de soluciones generadas. Este criterio de parada es útil debido a que no depende del tipo de procesador ni del lenguaje de programación utilizado. Adicionalmente, permite hacer comparaciones con otros autores quienes usan este mismo criterio.

Parámetros

El procedimiento descrito hace uso de 6 parámetros: α, β, T_{max} (temperatura máxima), T_{min} (temperatura mínima), ρ y valor del criterio de parada. Sin embargo sólo se requiere ajustar 3 parámetros para encontrar el mejor funcionamiento del algoritmo.

El valor del número de soluciones a generar (criterio de parada), por cuestiones de comparación es 50000, aunque se generan registros adicionales con 1000 y 5000 soluciones.

Con respecto a los parámetros α, β, T_{max} y T_{min}, se transforman en dos parámetros:

$$p_{\max} = \alpha \times e^{\beta x \frac{-1}{T_{\max}}} \qquad (11)$$

$$p_{\min} = \alpha \times e^{\beta x \frac{-1}{T_{\min}}} \qquad (12)$$

Así, T_{max} y T_{min} tendrán valores fijos y, α y β se despejan de las ecuaciones anteriores. P_{max} se puede interpretar como la probabilidad de aceptación asignada, al inicio de la búsqueda ($T = T_{max}$), a una solución con makespan una unidad por encima de la mejor solución conocida, y P_{min} se puede interpretar como la probabilidad de aceptación asignada, al final de la búsqueda ($T - T_{min}$), a una solución con makespan una unidad por encima de la mejor solución conocida. Adicionalmente, los nuevos parámetros tienen significados más fáciles de comprender por usuarios que no tengan conocimiento en métodos

heurísticos. Los parámetros P_{max}, P_{min} y P serán ajustados en la sección de resultados.

Resultados

Para realizar el análisis se tuvieron en cuenta los conjuntos de instancias pertenecientes a la librería PSPLIB [39], disponibles en la dirección electrónica http://129.187.106.231/psplib/. De dicha librería se tomó un conjunto de instancias de 30 actividades, cada una requiere cuatro tipos de recursos. Además, en dicha librería se encuentran publicadas las soluciones óptimas de dichas instancias.

La evaluación de los algoritmos será realizada teniendo en cuenta como criterio de parada el número de soluciones generadas. Este criterio de parada es utilizado por [40] para comparar los resultados de varios autores. La razón para usar este criterio de parada es que los resultados no dependen ni del lenguaje de programación utilizado ni del procesador; además, el tiempo computacional utilizado para generar una solución es, en general, poco variable en la mayoría de las heurísticas según lo afirmado por Kolisch y Hartmann.

Los resultados son comparados con base en la calidad (desviación porcentual con respecto a la solución óptima) promedio obtenida con 1000, 5000 y 50000 soluciones.

La experimentación se divide en cuatro fases: en la primera se hace una comparación sobre el desempeño del algoritmo utilizando uno, dos y tres vecindarios, en la segunda se evalúa el uso del método Justificación en el algoritmo, en la tercera se realiza el ajuste de parámetros, y en la cuarta se realiza la comparación del desempeño del algoritmo con el de otros investigadores a nivel internacional.

Uso de los vecindarios

Para evaluar la necesidad de usar en el procedimiento los tres vecindarios descritos, se realizaron diferentes ejecuciones con todas las posibles combinaciones de los tres vecindarios (el vecindario 1 representa el vecindario denominado Intercambios, el vecindario 2 representa la Inserción hacia adelante, y el vecindario 3 representa la Inserción hacia atrás). La tabla 1 presenta los resultados obtenidos en dicha experimentación con 50.000 soluciones generadas. En dicha tabla se presenta el valor promedio de 5 muestras, el valor mínimo, el valor máximo y el coeficiente de variación.

Tabla 1 Desempeño del algoritmo con diferentes vecindarios y 50000 soluciones generadas

Vecindarios	*Promedio*	*Mínimo*	*Máximo*	*Coef. Var.*
1	0,54	0,49	0,57	0,08
2	0,34	0,33	0,36	0,03
3	0,25	0,24	0,27	0,05
1 y 2	0,20	0,19	0,22	0,06
1 y 3	0,25	0,24	0,27	0,05
2 y 3	0,19	0,17	0,21	0,09
1, 2 y 3	0,17	0,15	0,19	0,10

De los resultados anteriores se puede observar que, cuando se usa sólo un vecindario, el vecindario de inserción hacia atrás presenta el mejor desempeño en cuanto a valor promedio, mínimo y máximo, y el vecindario de intercambios presenta el menor desempeño. Sin embargo, en cuanto al coeficiente de variación, el vecindario de intercambios es el mejor.

Cuando se usan dos vecindarios se puede observar algo similar a lo encontrado con un vecindario. El mejor desempeño del algoritmo se obtiene cuando se utilizan los vecindarios de inserción hacia adelante e inserción hacia atrás, aunque su coeficiente de variación es el mayor. Y, por otro lado, el menor desempeño se obtiene al utilizar los vecindarios de intercambio e inserción hacia

atrás. En general, es mejor utilizar dos vecindarios en lugar de uno sólo.

También es importante notar que, aunque los vecindarios de intercambio e inserción hacia adelante son los de menor desempeño individual, la combinación entre ellos mejora significativamente el desempeño del algoritmo, similarmente pasa con las otras combinaciones, aunque en menor medida. Esto puede deberse a que dichos vecindarios son complementarios y pocas de las soluciones pertenecientes a un vecindario, pertenecen a otro.

Al utilizar los tres vecindarios, en general, se mejoran los resultados obtenidos con dos vecindarios con respecto a las tres medidas de desempeño.

Observaciones similares fueron realizadas al utilizar otros valores para el criterio de parada.

Nuevamente, los mejores resultados en el desempeño del algoritmo con respecto al valor promedio, valor mínimo y valor máximo, implican un aumento en el coeficiente de variación. Sin embargo, esto no representa mayor riesgo debido a que los valores máximos obtenidos con los tres vecindarios y 50000 soluciones son, en general, menores a los valores mínimos obtenidos con otras combinaciones.

Uso del procedimiento justificación

Para evaluar la eficiencia del procedimiento Justificación, se ejecutó el algoritmo utilizando dicho procedimiento y sin usarlo. Los resultados son resumidos en la tabla 2.

De los resultados anteriores, es notable la mejora del desempeño del algoritmo cuando se usa el procedimiento Justificación, con respecto a cuando no lo usa. El uso de este procedimiento mejora el 100% de las veces los resultados obtenidos por el algoritmo, mejorando la desviación porcentual entre el 40% y el 70%.

Tabla 2 Desempeño del algoritmo utilizando el procedimiento Justificación

	# Soluciones	*Promedio*	*Máximo*	*Mínimo*	*Coef. Var.*
Sin justificación	1000	1,93	1,96	1,88	0,01
	5000	1,13	1,17	1,08	0,04
	50000	0,48	0,49	0,47	0,02
Con justificación	1000	1,14	1,22	1,07	0,05
	5000	0,61	0,68	0,56	0,07
	50000	0,17	0,19	0,15	0,10

Ajuste de parámetros

Para realizar el ajuste de los parámetros P_{max}, P_{min} y P, se realizó un diseño experimental completo con los siguientes niveles para cada parámetro: $P_{max} \in \{0,35; 0,5; 0,63\}$, $P_{min} \in \{0,01; 0,05; 0,1\}$ y $P \in \{0,5; 0,7; 0,9\}$. Para cada combinación de valores de los parámetros se realizaron cinco muestras para un total de 135 corridas del algoritmo.

Con los resultados obtenidos se realizó un Análisis de Varianza Multifactor (ANOVA), con los tres parámetros en estudio como factores y el desempeño del algoritmo como variable dependiente, para evaluar la influencia de los valores de los parámetros. En la Tabla 3 se presentan los valores p para cada factor (parámetro), de los que se puede concluir que los parámetros P_{min} y P no son estadísticamente

significativos a un nivel de confianza del 95% para explicar la variabilidad en los datos, así como las interacciones entre los parámetros.

Debido a que los parámetros P_{min} P y no son estadísticamente significativos, se realizó una prueba adicional utilizando una probabilidad de aceptación fija durante toda la ejecución del algoritmo, es decir, con $\rho = 1$. Los resultados confirman la influencia de P_{max} en el desempeño del algoritmo, sin embargo, indican que dicha probabilidad debe variar durante la búsqueda para obtener un mayor desempeño. Lo anterior significa que los parámetros P_{min} y ρ son necesarios, aunque sus valores no sean relevantes.

Tabla 3 ANOVA Multifactor

Factor		*Valor p*
Efectos principales	p_{min}	0,1809
	p_{max}	0,0002
	ρ	0,2653
Interacciones	p_{min}, p_{max}	0,5500
	p_{min}, ρ	0,5193
	p_{max}, ρ	0,4971

Se realizaron pruebas adicionales para determinar valores específicos de los parámetros en estudio. De acuerdo con las muestras, los valores más apropiados de los parámetros P_{min} y ρ son 0,01 y 0,7, respectivamente. Para el parámetro P_{max} se encontró que los mejores resultados se obtienen con valores alrededor de 0,9; sin embargo para valores cercanos a 0,7 se obtiene resultados con menor variabilidad.

Comparación con otros algoritmos

En la tabla 4, basada en [40], se recopilan los resultados obtenidos por otros investigadores, a nivel internacional, que trabajan los problemas de 30 actividades de la librería PSPLIB.

De acuerdo con la tabla 4, el algoritmo desarrollado en esta investigación se ubica tal como se enuncia a continuación: Para 1000 schedules, en el puesto número 22, entre 29 posiciones, con una diferencia de 0.11% respecto a la posición anterior. Para 5000 schedules, en el puesto número 19, entre 29 posiciones, con una diferencia de 0.05% respecto a la posición anterior. Para 50000 schedules, en el puesto número 13, entre 21 posiciones, con una diferencia de 0.01% respecto a la posición anterior.

Conclusiones

En este estudio se diseñó un algoritmo heurístico híbrido combinando diferentes estrategias de búsqueda. Este algoritmo usa múltiples estructuras de vecindarios como estrategia de intensificación, además de procedimientos como Justificación y una estrategia de Adaptación. Como estrategias de diversificación, se permite aceptar soluciones de no mejora basándose en las ideas del Recocido Simulado.

Con respecto a los parámetros, se propuso una transformación a los parámetros originales del Recocido Simulado, haciéndolos más claros para los usuarios. Al utilizar técnicas estadísticas como ANOVA se encontró que los parámetros P_{min} y ρ no son estadísticamente significativos para explicar la variabilidad de los datos.

Las pruebas realizadas comprueban la utilidad del método de Justificación en la solución del RCPSP. Así mismo, se halló que al utilizar más vecindarios, se mejora el desempeño del algoritmo; sin embargo la variabilidad de la solución aumenta.

El algoritmo diseñado presenta un buen desempeño en comparación con los resultados obtenidos por otros investigadores. Esto se refleja en las soluciones obtenidas, las cuales superan las de otros algoritmos con 50000 soluciones como criterio de parada. Sin embargo, se puede observar que para otros valores del criterio de parada, el algoritmo presentado obtiene un menor desempeño. Lo anterior indica un grado de convergencia lento en las primeras etapas de la búsqueda.

Tabla 4 Comparación con otros investigadores

Algoritmo	*Autor*	*Número máximo de soluciones*		
		1.000	*5.000*	*50.000*
GA, TS – path relinking	Kochetov, Stolyar	0,10	0,04	0,00
Scatter Search – FBI	Debels et al.	0,27	0,11	0,01
GA – hybrid, FBI	Valls et al.	0,27	0,06	0,02
GA – FBI	Valls et al.	0,34	0,20	0,02
GA – forw.-backw., FBI	Alcaraz et al.	0,25	0,06	0,03
GA – forw.-backward	Alcaraz, Maroto	0,33	0,12	-
Sampling – LFT, FBI	Tormos, Lova	0,25	0,13	0,05
TS – activity list	Nonobe, Ibaraki	0,46	0,16	0,05
Sampling – LFT, FBI	Tormos, Lova	0,30	0,16	0,07
GA – self-adapting	Hartmann	0,38	0,22	0,08
GA – activity list	Hartmann	0,54	0,25	0,08
Sampling – LFT, FBI	Tormos, Lova	0,30	0,17	0,09
TS – activity list	Klein	0,42	0,17	-
sampling – random, FBI	Valls et al.	0,46	0,28	0,11
VNS – SA – FBI	Esta investigación	0,97	0,49	0,12
SA – activity list	Bouleimen, Lecocq	0,38	0,23	-
GA – late join	Coelho, Tavares	0,74	0,33	0,16
sampling - adaptive	Schirmer	0,65	0,44	-
TS – Schedule scheme	Baar et al.	0,86	0,44	-
sampling - adaptive	Kolisch, Drexl	0,74	0,52	-
GA – random key	Hartmann	1,03	0,56	0,23
sampling - LFT	Kolisch	0,83	0,53	0,27
sampling - global	Coelho, Tavares	0,81	0,54	0,28
sampling - random	Kolisch	1,44	1,00	0,51
GA – priority rule	Hartmann	1,38	1,12	0,88
sampling – WCS	Kolisch	1,40	1,28	-
sampling – LFT	Kolisch	1,40	1,29	1,13
sampling – random	Kolisch	1,77	1,48	1,22
GA – problema space	Leon, Ramamoorthy	2,08	1,59	-

Referencias

1. A. Mingozzi, V. Maniezzo, S. Ricciardelli, L. Bianco. "An exact Algorithm for the Resource Constrained Project Scheduling Problem Based on a New Mathematical Formulation". *Management Science*. Vol. 44. 1998. pp. 714-729.
2. L. Tseng, S. Chen. "A hybrid metaheuristic for the resource-constrained project scheduling problem". *European Journal of Operational Research*. Vol. 175. 2006. pp. 707-721.
3. V. Valls, F. Ballestín, S. Quintanilla. "Justification and RCPSP: a technique that pays". *European Journal of Operational Research*. Vol. 165. 2005. pp. 375-386.
4. B. Abbasi, S. Shadrokh, J. Arkat. "Bi-objective resource-constrained project scheduling with robustness and makespan criteria". *Applied mathematics and computation.* Vol. 180. 2006. pp. 146-152.
5. J. Blazewicz, J. Lenstra, A. Rinnooy Kan. "Scheduling Projects Subject to Resource Constraints: Classification and Complexity". *Discrete Applied Mathematics*. Vol. 5. 1983. pp. 11-24.
6. D. Merkle, M. Middendorf, H. Schmeck. "Ant colony optimization for resource-constrained project scheduling". *IEEE Transactions on Evolutionary Computation*. Vol. 6. 2002. pp. 333-346.
7. E. Balas. "Project Scheduling with Resource Constraints". In: E. M. L. Beale (editor). *Application of Mathematical Programming Techniques*. Ed. Elsevier. New York. 1970. pp. 187-200.
8. L. Schrage. "Solving Resource-Constrained Network problems by Implicit Enumeration – Non-preemptive Case". *Operations Research*. Vol. 18. 1971. pp. 225-235.
9. S. Gorenstein. "An Algorithm for Project Sequencing with Resource Constraints". *Operations Research*. Vol. 20. 1972. pp. 835-850.
10. M. Fisher. "Optimal Solution of Scheduling Problems Using Lagrange Multipliers. Part I". *Operations Research*. Vol. 21. 1973. pp. 1114-1127.
11. J. Patterson, W. Huber. "A Horizon-Varying, ZERO-One Approach to Project Scheduling". *Management Science*. Vol. 20. 1974. pp. 990-998.
12. J. Patterson, G. Roth. "Scheduling a Project under Multiple Resource Constraints: a Zero-One Programming Approach". *AIIE Transactions*. Vol. 8. 1976. pp. 449-455.
13. F. Talbot, J. Patterson. "An Efficient integer programming Algorithm with Network Cuts for Solving Resource-Constrained Scheduling problems". *Management Science*. Vol. 24. 1978. pp. 1163-1174.
14. E. Demeulemeester, W. Herroelen. "A Branch and Bound Procedure for the Multiple Resource-Constrained project Scheduling Problem". *Management Science*. Vol. 38. 1992. pp. 1803-1818.
15. W. Simpson, J. Patterson. "A multiple-tree search procedure for the resource-constrained project scheduling problem". *EJOR.* Vol. 89. 1996. pp. 525-542.
16. E. Demeulemeester, W. Herroelen. "New benchmark results for the resource-constrained project scheduling problem". *Management Science*. Vol. 43. 1997. pp. 1485-1492.
17. A. Sprecher, S. Hartmann, A. Drexl. "An exact algorithm for project scheduling with multiple modes". *OR Spektrum*. Vol. 19. 1997. pp. 195 - 203.
18. P. Brucker, S. Knust, A. Schoo, O. Thiele. "A branch & bound algorithm for the resource-constrained project scheduling problem". *European Journal of Operacional Research*. Vol. 107. 1998. pp. 272-288.
19. A. Sprecher. "Scheduling resource-constrained projects competitively at modest resource requirements". *Management Science*. Vol. 46. 2000. pp. 710-723.
20. J. Damay, A. Quilliot, E. Sanlaville. "Linear programming based algorithms for preemptive and non-preemptive RCPSP". *European Journal of Operational Research*. Vol. 182. 2007. pp. 1012-1022.
21. M. Sabzehparvar, S. Seyed Hosseini. "A mathematical model for the multi-mode resource-constrained project scheduling problem with mode dependent time lags". *The Journal of Supercomputing*. Vol. 44. 2008. pp. 257-273.
22. R. Martí. "Procedimientos metaheurísticos en optimización combinatoria". *Matemátiques*. Vol. 1. 2003. pp. 3-62.
23. Z. Michalewicz, D. Fogel. *How to solve it: modern heuristics*. Ed. Springer. New York. 2002. pp. 117-125.
24. S. Kirkpatrick, C. D. Gelatt, M. P. Vecchi. "Optimization by simulated annealing". *Science,* Vol. 220. 1983. pp. 671-680.
25. N. Metropolis, A. Rosenbluth, A. Teller, E. Teller. "Equations of state calculations by fast computing machines". *The journal of chemical physics*. Vol. 21. 1953. 1087-1092.

26. L. Moreno, J. Rivera, J. Diaz, G. Peña. "Análisis comparativo entre dos algoritmos heurísticos para resolver el problema de planeación de tareas con restricción de recursos (RCPSP)". *Dyna*. Vol. 74. 2007. pp. 171-183.

27. K. Bouleimen, H. Lecocq. "A new efficient simulated annealing algorithm for the resource-constrained project scheduling problem and its multi mode version". *European Journal of Operational Research*. Vol. 149. 2003. pp. 268-281.

28. J. Cho, Y. Kim. "A simulated annealing algorithm for resource constrained project scheduling problems". *Journal of the Operational Research Society.* Vol. 48. 1997. pp. 736-744.

29. F. Boctor. "Resource-constrained project scheduling by simulated annealing". *International Journal of Production Research*. Vol. 34. 1996. pp. 2335-2351.

30. J. Lee, Y. Kim. "Search heuristics for Resource Constrained Project Scheduling". LG-EDS Corporation y Korea Advanced Institute of Science and Technology. *Journal of the Operational Research Society*. Vol. 47. 1996. pp. 678-689.

31. C. Koulamas, S. Anthony, R. Jean. "A survey of simulated annealing applications to operations research problems". *Omega*. Vol. 22. 1994. pp. 41-56.

32. P. Hansen, N. Mladenovic. "Variable Neighborhood Search". *Search Methodologies. Introductory tutorials in optimization and decision support techniques*. Ed. Springer. New York. 2005. pp. 221-238.

33. K. Fleszar, K. Hindi. "Solving the resource-constrained project scheduling problem by a variable neighbourhood search". *European Journal of Operational Research*. Vol. 155. 2004. pp. 402-413.

34. J. Wiest. "Some properties of schedules for large projects with limited resources". *Operations Research*. Vol. 12. 1964. pp. 395-418.

35. D. Debels, B. De Reyck, R. Leus, M. Vanhoucke. "A hybrid scatter search / Electromagnetism meta–heuristic for project scheduling". *European Journal of Operational Research.* Vol. 169. 2006. pp. 638-653.

36. E. Buffa, S. Edwood S. Sarin, K. Rakesh. *Administración de la producción y de las operaciones*. Ed. Limusa. México D. F. 1992. pp. 381-406.

37. T. Baar, P. Brucker, S. Knust. "Tabu-Search Algorithms and lower bounds for the Resource-Constrained Project Scheduling Problem". In: S. Voss, S. Martello, I. Osman, C. Roucairol. (editors) *Meta-heuristics: Advances and trends in local search paradigms for optimization.* Ed. Kluwer Academic Publishers. Boston. 1998. pp. 1-18.

38. J. Stinson, E. Davis, B. Khumawala. "Multiple Resource-Constrained Scheduling Using Branch and Bound". *AIIE Transactions.* Vol. 10. 1978. pp. 252.259.

39. R. Kolisch, C. Schwindt, A. Sprecher. "Benchmark instances for project scheduling problems". *Project Scheduling – Recent Models, Algorithms and Applications.* J. Weglarz (editor). Kluwer Academic Publishers. Boston. 1999. pp. 197-212.

40. R. Kolisch, R. Hartmann. "Experimental Investigation of Heuristics for Resource-Constrained Project Scheduling: An Update". *European Journal of Operational Research*. Vol. 174. 2006. pp. 23-37.