Inteligencia Artificial

Estado del Arte: Resource Constrained Project Scheduling Problem

Matías Valdés

28 de julio de 2012

Evaluación

Resumen (5 %):	_____
Introducción (5 %):	_____
Definición del Problema (10 %):	_____
Estado del Arte (35 %):	_____
Modelo Matemático (20 %):	_____
Conclusiones (20 %):	_____
Bibliografía (5 %):	_____
Nota Final (100 %):	_____

Resumen

El RCPSP es uno de los problemas de optimización más estudiados. Consiste en encontrar la mejor programación de actividades de un proyecto dadas restricciones de precedencia entre las actividades y restricciones de uso de recursos limitados para la realización de las actividades. Dada la caracterización del problema la búsqueda de soluciones rápidamente se vuelve difícil, siendo necesario el uso de técnicas heurísticas para instancias de al rededor de 30 actividades. En los últimos años se han presentado algoritmos híbridos, que complementan el uso de meta-heurísticas para búsquedas globales con técnicas que mejoran las soluciones localmente buscando en las vecindades. En el presente informe se presenta una definición del problema, un estado del arte, un modelo matemático y se aplica la técnica de Simulated Annealing para su resolución y experimentación.

1. Introducción

El propósito del presente trabajo es estudiar el Problema de Planificación de Proyectos con Recursos Restringidos o *Resource Constrained Project Scheduling Problem, (RCPSP)*. Este problema es de gran interés ya que está presente en variados dominios donde se debe planificar la ejecución de un proyecto con determinadas actividades y recursos limitados. El problema surge al caracterizar el proyecto compuesto por un conjunto de actividades las que poseen restricciones de precedencia sobre otras, y además estas actividades requieren de recursos que están limitados para realizarse, y se busca (generalmente) reducir el tiempo en que el proyecto es completado. Esta simple caracterización impone fuertes restricciones sobre el problema situándolo dentro de los problemas de mayor complejidad a medida que el problema crece, además a medida que se

intenta representar escenarios mas realistas, se imponen aún mas restricciones. La importancia en encontrar algoritmos que resuelvan el problema es tal que una mejor solución puede implicar importantes ahorros en tiempo y dinero según el dominio al que se aplique.

Los objetivos a lograr con este estudio son, investigar el problema RCPSP, conocer las características principales, así como las técnicas heurísticas y meta heurísticas aplicadas para resolverlo. Se realiza una implementación concreta para Simulated Annealing y se realiza una experimentación para evaluar el desempeño de la implementación, así como para observar en la práctica como las variaciones de los parámetro influyen en el proceso de búsqueda.

El trabajo primero define el problema en la sección 2, presentando además algunas variaciones, luego presenta el Estado del Arte en la sección 3 y presenta un modelo en la sección 4. Finalmente se entregan algunas conclusiones en la sección 9.

2. Definición del Problema

El Problema de Planificación de Proyectos con Recursos Restringidos o *Resource Constrained Project Scheduling Problem, (RCPSP)* es un problema de optimización combinatoria muy estudiado y, como se verá en la sección 3, se ha intentado solucionar desde diversos enfoques desde métodos de búsqueda completa, usando heurísticas, meta-heurísticas o métodos híbridos.

El objetivo de los algoritmos es encontrar una planificación de las tareas, es decir encontrar el tiempo en que se inician y de que forma se usarán los recursos que necesita.

Algunos estudios utilizan enfoques probabilísticos sobre la duración de las tareas o el proyecto, pero la mayoría de los estudios se definen de manera determinista, en donde el tiempo es una división discreta.

También se consideran variantes en que las actividades pueden ser interrumpidas *(preemptive)* o no *(non-preemptive)*, utilizándose como condiciones del problema real o como relajaciones utilizadas para encontrar límites inferiores *Lower Bound (LB)* para comparación del algoritmo.

Es importante señalar que la forma usual de comparación entre el algoritmo y el Lower Bound es en base a la diferencia relativa:

$$\frac{C_{max} - LB}{LB}$$

.

De esta forma, mientras mejor es la cota (mas cercana al óptimo), mas precisa será la comparación de los algoritmos mediante sus resultados.

El caso usual de RCPSP consiste en un único proyecto que posee un conjunto V de n tareas o actividades que deben ser realizadas y un conjunto R de K recursos que las actividades requieren para realizarse, los cuales poseen las siguientes características.

- Las actividades poseen restricciones de precedencia sobre el orden que deben ser realizadas. Una actividad j no puede comenzar hasta que sus predecesoras inmediatas hayan finalizado. Estas pueden ser representadas por un grafo dirigido $G = (V, A)$ de actividades en el nodo que se asume es acíclico, donde V es el set de actividades y A el set de relaciones de precedencia.
- Se asumen dos actividades *dummy* artificiales en V, sin duración ni recursos. Una actividad j_0 que representa el inicio del proyecto, y una actividad j_{n+1} que representa el final.
- Para realizarse, cada actividad j requiere $r_{j,k}$ unidades del recurso $k \in R$.
- Cada recurso k tiene capacidad R_k limitada en cada periodo. Se dice que el recurso es renovable si su capacidad R_k es constante por cada periodo de asignación, es decir, cuando una actividad j que requiere $r_{j,k}$ termina, el recurso k se devuelve y mantiene su capacidad constante.

- Cada actividad j posee duración p_j, la cual una vez iniciada no puede ser interrumpida ni sus recursos liberados.
- Todos los parámetros se asumen enteros no negativos.

El objetivo es minimizar el tiempo en que se completa el proyecto o *makespan*, es decir la diferencia entre el tiempo final y el tiempo inicial, manteniendo las restricciones sobre precedencia de actividades y disponibilidad de recursos. Este tiempo suele ser descrito como $C_{max} = max_{i=1}^{n} C_i$, donde $C_i = S_i + p_i$ es el tiempo en que se completa la actividad i (S_i el tiempo (instante) en que inicia y p_i el tiempo que demora en completarse).

Una instancia de ejemplo es la que se muestra en la figura 1. Las restricciones de precedencias van dadas por el grafo, y por cada actividad (nodo) se requieren unidades de 3 recursos para completar la actividad en cierto tiempo, esto último se denomina modo, el que se anota $p_j|r_{j,1}, .., r_{j,k}$

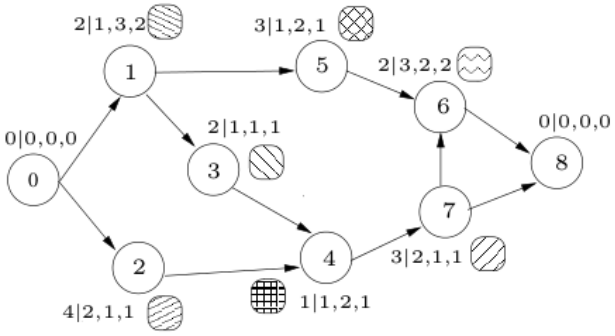

Figura 1: Instancia de ejemplo de actividades con restricciones de precedencia y recursos.

Y una posible solución de las actividades programadas y que satisface los requerimientos se muestra en la figura 2.

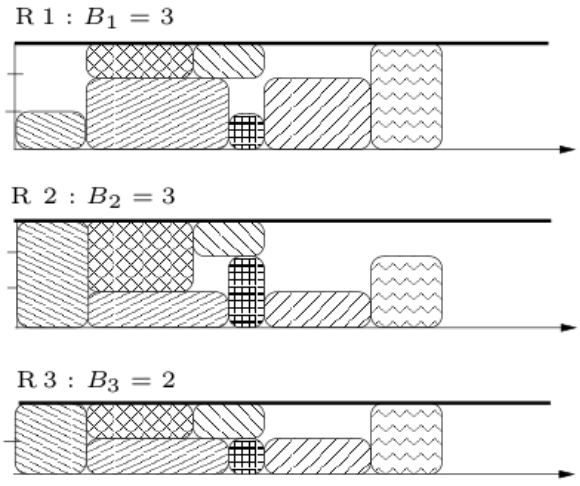

Figura 2: Solución factible de la instancia

2.1. Variante en el Modo de la Actividad

La variante multi-modal o múltiple *MRCPSP* consiste en que para cada actividad j existe más de una combinación posible de recursos para realizarse, variando además el costo o tiempo requerido. Esto puede ser utilizado para definir distintos tipos de recursos.

Los algoritmos de búsqueda completa generalmente se basan en seleccionar una actividad posible dada las restricciones de precedencia y luego dentro de la actividad seleccionar modos posibles. Otros algoritmos como el propuesto en [11] buscan seleccionar un modo y luego podar alternativas de actividades. Sin embargo para los casos multimodo se prefiere el uso de heurísticas [5] dada la complejidad alcanzada con pocas actividades y pocos modos.

2.2. Variante en la Generalización de Precedencia

En el RCPSP típico la restricción de precedencia entre una actividad j y una actividad i predecesora se entiende de tipo *Finish to Start* con parámetro cero: $FS = 0$, es decir la actividad i anterior debe finalizar para que la actividad j pueda comenzar e inicia inmediatamente cuando esto ocurre.

Estas relaciones se puede generalizar obteniendo los cuatro casos: *SS, SF, FS y FF*, además de agregar como parámetro un *Lag* o tiempo de espera mínimo, o dos parámetros, un lag mínimo y un lag máximo. Estos problemas se identifican como RCPSP/max.

Estos parámetros suelen representarse como:

$$EF_i = \text{El menor tiempo final para la actividad } i$$

$$ES_i = \text{El menor tiempo inicial para la actividad } i$$

$$LF_i = \text{El mayor tiempo final para la actividad } i$$

$$LS_i = \text{El mayor tiempo inicial para la actividad } i$$

2.3. Variante en el uso del recurso

Esta variación permite definir tipos en los recursos de acuerdo a su uso. Como se mencionó en la definición del problema los recursos renovables (R) son aquellos en que su capacidad R_k es constante por cada periodo de asignación, es decir su disponibilidad varía según los requerimientos, pero una vez finalizadas las actividades que lo ocupaban, su disponibilidad vuelve a ser la inicial.

Los recursos no renovables (N) son los que una vez que las actividades que lo ocupaban terminan, no vuelven a restaurar su disponibilidad inicial, por lo que se van consumiendo conforme se ejecutan las actividades del proyecto.

Los recursos doblemente restringidos (D), son los recursos que se encuentran limitados tanto durante el periodo que la actividad los ocupa, como por un total asociado al proyecto. Por ejemplo un presupuesto que no puede superar un máximo pero además es decrementado por cada uso de la actividad j.

2.4. Variantes dinámicas

El caso usual de RCPSP se considera estático desde el punto de vista de que entrega una solución que no admite ser cambiada mientras se desarrollan las actividades, a diferencia de algoritmos dinámicos que pueden adaptarse a cambios en las restricciones de precedencia o de recursos.

Esto puede lograrse realizando una re-programación global cada vez que un cambio ocurre, o actualizando un esquema inicial estático realizando búsquedas locales.

Para el caso dinámico en que es necesaria la inserción de actividades se estudia un mecanismo de inserción en [2]. Este mecanismo también es utilizado para la formulación de soluciones en casos estáticos utilizando Tabu Search.

3. Estado del Arte

A mediados de los años 60s se encuentran los primeros estudios relacionados, y ya en 1969 en el trabajo de Pritsker et al [19] se encuentran las primeras formalizaciones en un modelo matemático como un *job-shop scheduling* multiproyecto, en este caso se presenta un problema de los requerimientos de las actividades de los proyectos forman una matriz de $k \times ij$, donde k representa un recurso, i el proyecto y j la actividad. Las actividades también se encuentran restringidas por una matriz de precedencia y se presentan 3 funciones objetivos. La forma propuesta para su resolución es en base a programación lineal utilizando variables binarias.

Ya en 1972 Herroelen publica un estado del arte en RCPSP [12], pero posteriormente no se encuentra mucho avance (en cuanto a trabajos publicados) lo que se puede atribuir a la falta de interés en Inteligencia Artificial a principios de los años 70s.

Luego en En 1983 Blazewicz et al [4] caracterizan el problema como una generalización de *job-shop scheduling* y que cae en la categoría de los problemas $\mathcal{NP}$-hard. Si bien ya se habían aplicado heurísticas, este trabajo justifica y crea mayor interés por el uso de heurísticas para la resolución de estos problemas, para instancias relativamente grandes.

En [8] se presenta una clasificación de los problemas y una notación basada en el esquema utilizado para problemas de machine scheduling, esto es basada en el esquema $\alpha|\beta|\gamma$, que definen el ambiente del recurso, las características de las actividades, y la función objetivo, respectivamente. De esta forma el RCPSP típico se anota como $PS|prec|C_{max}$

3.1. Métodos Completos

3.1.1. Programación Lineal

En [6] se presenta un método de programación lineal basado en generación de columnas y una forma para para calcular límites inferiores basado en dos métodos, el primero utiliza técnicas de propagación de restricciones y el segundo se basa en una formulación de programación lineal. En general suelen utilizarse para resolver instancias relativamente pequeñas (al rededor de 30 o menos actividades) o encontrar Cotas inferiores para instancias mayores utilizando casos relajados.

3.1.2. Branch & Bound

Esta técnica basada en backtracking hace uso de una representación de árbol para el espacio de búsqueda el que se va formando de forma dinámica mientras se enumeran las distintas combinaciones y podando el árbol para aquellas ramas que generan soluciones que se alejan de la solución óptima.

En esta línea se pueden encontrar aplicaciones al RCPSP desde el trabajo Doctoral de TJR Johnson en 1967. En [18] se propone una metodología para MRCPSP basada en búsqueda en profundidad *(deep-first)* que se compone de una primera fase de iniciación, en que las actividades son reordenadas de acuerdo a reglas de priorización de despacho, los modos de actividad son ordenados de manera similar y a las actividades son asignadas a un tiempo de completitud del proyecto en base a la ruta crítica. Según las convenciones usadas en el ordenamiento se partirá de una solución inicial y se guiará la búsqueda. Luego una segunda fase de enumeración que genera el espacio de solución y evalúa las soluciones parciales.

La búsqueda es estructurada como un Árbol de Precedencia *(Precedence Tree)* que resulta de la asignación de las actividades a los tiempos de completitud de precedencias posibles.

Cuando una actividad j es asignada a un tiempo de completitud posible, la lista de actividades predecesoras posibles aún no programadas es actualizada para incluir los sucesores inmediatos de j. Luego se intenta asignar la siguiente actividad de la lista actualizada. Si como resultado la inserción no es posible dentro de los límites superiores para esta actividad, el algoritmo genera el salto hacia atrás por la rama del último nodo creado. Luego se intenta insertar otra actividad y así sucesivamente. Una vez seleccionada una actividad se selecciona un modo. Para el objetivo de minimizar el *makespan*, cuando una nueva solución es encontrada, los límites superiores basados en tiempo son ajustados para ser una unidad menos.

En [7] Brucker et al proponen un mecanismo en para RCPSP en que se parte desde conjunción de pares de actividades dadas por el grafo de precedencia y un conjunto de disyunciones dadas por las restricciones de los recursos. Luego o introduce restricciones de paralelismo sobre la lista de actividades actividades o inserta actividades en paralelo. Para esto introduce el concepto de selección inmediata. Propone además un método de generación de columnas para resolver mediante programación lineal.

En [11] Hartmann y Drexl revisan y comparan diversos métodos completos aplicados a MRCPSP. Comparan los distintos esquemas propuestos para la enumeración, centrándose en tres basados en el esquema de *(Precedence Tree)*. Compara con el método *Mode and Delay Alternatives* que ajusta los límites de tiempo por cada nivel g del árbol a un instante fijo en el cual las actividades pueden comenzar. Luego una actividad j es elegible en un instante t_g si todas sus predecesoras $i \in P_j$ poseen un tiempo de finalización $f_i < t_g$. Además propone el método *Mode and Extensión Alternatives* que realiza extensiones de la rama al seleccionar un modo nuevo para la actividad en curso.

3.2. Heurísticas

3.2.1. Schedule generation schemes

El esquema de generación de secuencias o *Schedule generation schemes (SGS)* consiste en construir una programación posible paso a paso extendiendo una programación parcial inicial. Pueden ser seriales si en cada paso el incremento es en base a las actividades o paralela si es en base a periodos de tiempo. El objetivo es construir una secuencia de $n + 1$ tiempos de inicio o finalización.

3.2.2. Serial SGS

El caso serial consiste de $g = \{1, .., n\}$ etapas. En cada etapa se selecciona una actividad y se programa para que comience en el tiempo más temprano sujeto a las restricciones de precedencia y recursos. Por cada secuencia g hay dos conjuntos, un conjunto S_g de actividades programadas y un conjunto D_g de actividades elegibles. Además por cada etapa se calcula $F_g = \{F_j | j \in Sg\}$, el conjunto de tiempos de finalización mas temprana de las actividades seleccionadas en D_g.

El algoritmo comienza incluyendo la actividad inicial. Por cada etapa g calcula S_g, D_g y F_g, selecciona una $j \in D_g$, determina EF_j, el tiempo de finalización mas temprano de j compatible con las precedencias, determina F_j, el tiempo de finalización mas temprano de j compatible con precedencias y recursos dentro de $[EF_j, LF_j]$ (LF_j calculado como el tiempo de finalización más tardío usando recursión por las restricciones de precedencia en base a un parámetro de horizonte de tiempo T), Incorpora j a S_g. Luego finaliza insertando la actividad final con duración 0. El tiempo de finalización de esta actividad corresponde a la duración de la secuencia. La complejidad del algoritmo es de orden $\mathcal{O}(n^2 \cdot K)$.

3.2.3. Parallel SGS

En el caso paralelo, por cada etapa g se posee un tiempo t_g, un conjunto $C_g = \{j \in V \mid F_j \leq t_g\}$ que posee la lista de actividades que se han completado hasta el periodo t_g; un conjunto

$A_g = \{j \in V \mid F_j - p_j \leq t_g < F_j\}$ que posee las actividades que se encuentran activas durante t_g; y el conjunto D_g de actividades que pueden ser comenzadas en t_g respetando las restricciones de precedencia y recursos.

El algoritmo inicia en $g = 0$, con $t_g = 0$. Se itera mientras $|A_g \cup C_g| \leq n$. Se incrementa $g = g+1$, y $t_g = min\{F_j\}$, $j \in A_g$.Se calcula C_g, A_g y D_g. Luego mientras $D_g \neq 0$ se selecciona una actividad j, se actualiza $F_j = t_g + p_j$ y se actualizan $A_g y D_g$. Se finaliza igual que en el caso serial incorporando la actividad final. La complejidad del caso paralelo también es de orden $\mathcal{O}(n^2 \cdot K)$.

3.2.4. Priority Rules

Para seleccionar las actividades elegibles se requieren reglas de prioridad que corresponderán a funciones para asignar un valor a cada actividad, sujeta a criterios de extremos máximo y mínimo. De esta forma se mapea cada $j \in D_g$ a un valor $v(j)$.

Las reglas pueden ser *locales* o *globales*, si en calcular el valor consideran solo la actividad en curso u otras actividades, respectivamente; o también pueden ser *estáticas* o *dinámicas*, según se mantengan constantes durante todo el algoritmo o varíen. En [15] se puede encontrar una clasificación de reglas de prioridad.

3.2.5. X-Pass Methods

En base a SGS y las reglas de prioridad se construye un algoritmo heurístico específico, si el algoritmo construye solo una programación de actividades se denomina *single pass method*. Si es más de una, la heurística se denomina *multi pass method*.

3.3. Representaciones

Generalmente los métodos meta-heurísticos utilizan representaciones y no modifican concretamente la programación de actividades. Para ello existen principalmente las siguientes representaciones: *Activity List*, *Random Key* y *Priority Rule*

Activity List Representation utiliza una secuencia de actividades posible en cuanto a las restricciones definida como:

$$\lambda = \langle j_1, j_2, ..., j_n]$$

Diversas operaciones sobre la lista han sido propuestas, generalmente que intercambian dos actividades de lugar, como por ejemplo las propuestas en [3].

Random Key Representation utiliza la siguiente lista:

$$\rho = (r_1, r_2, ..., r_n)$$

y que asigna un valor real r_j a la posición de la actividad j. Luego en ambos SGS, serial y paralelo se utilizan estos valores para seleccionar la actividad con el valor que indique la mayor prioridad. Estos valores generalmente son iniciados con valores aleatorios y se han propuesto distintas operaciones unarias sobre el manejo de las prioridades o sobre los parámetros de la función que asigna los valores aleatorios.

Priority Rule Representation utiliza la lista

$$\pi = \langle \pi_1, \pi_2, ..., \pi_n]$$

También ambos casos de SGS serial y paralelo utilizan esta representación para indicar que regla de prioridad π_j se utilizará para la programación de la tarea j.

3.4. Técnicas Meta-Heurísticos

3.4.1. Tabu Search (TS)

Baar et al en [3] proponen dos métodos de uso de Tabu Search, primero basado en la mejora de la solución mediante búsqueda local y saltar a otros espacios mediante eliminación de arcos; y segundo en la construcción de secuencias de esquemas de actividades donde los vecinos son generados ubicando actividades en paralelo o eliminando una relación de paralelismo. Además utiliza el método de generación de columnas y resolución mediante programación lineal presentado en [7] para comparar el algoritmo.

En [2] se utiliza el método de inserción de actividades en conjunto con Tabu Search. El método iterativamente selecciona actividades eliminándola desde la programación de actividades y luego reinsertándola. Las actividades seleccionadas, así como las predecesoras y sucesoras de su recurso son elementos de la lista tabú.

3.4.2. Simulated Annealing (SA)

Esta técnica está basada en como las partículas de un material al enfriarse van re-acomodándose en una configuración de mínima energía. Se asigna inicialmente un parámetro de temperatura. Comenzando por una solución inicial se construye una solución vecina modificando levemente la solución actual. Si la solución actual es mejor se acepta y la búsqueda continua en base a la nueva solución; si es peor, la solución puede ser aceptada en base a una probabilidad que depende de la magnitud del empeoramiento y de la temperatura. A medida que el algoritmo avanza,el valor de esta temperatura decrece por lo que se va reduciendo la probabilidad de aceptar soluciones peores.

Esta meta-heurística es usada en [20] donde se introduce el concepto de justificación para mejorar las programaciones generadas por un método SGS. La justificación consiste en juntar una programación ajustada en los límites hacia la izquierda, con una ajustada hacia la derecha, mediante un método presentado por Wiest en 1964 [21]. Luego verifican empíricamente que la justificación mejora las soluciones encontradas solo con SA.

3.4.3. Ant Colony Optimization (ACO)

A partir de la publicación de Ant Colony Optimization como meta-heurística [10] surgen después trabajos que la aplican a RCPSP como en [16] encontrando mejores resultados que con heurísticas anteriores.

3.4.4. Algoritmos Geneticos (GA)

GA considera conjuntos de poblaciones de soluciones, las que son producidas en base a una función de cruzamiento y/o alterando la población actual en base a una función de mutación. Luego de producidas la población mas ajustada sobrevive y logra pasar de generación. En [14] se presentan resultados empíricos que muestran que esta meta-heurística es la que mejor desempeño tiene.

3.5. Métodos Híbridos

Es relevante mencionar algunos trabajos que utilizan la combinación de meta-heurísticas para la búsqueda global en conjunto con otros métodos para optimizar la solución buscando localmente en las vecindades.

En [1] se utilizan algoritmos genéticos (GA) en conjunto con redes neuronales (NN). Ambos algoritmos comparten el espacio de las soluciones. La red neuronal selecciona soluciones desde las combinaciones encontradas por GA, las buenas soluciones encontradas por NN son agregadas

en la población de GA para futuras iteraciones. El estudio además demuestra empíricamente que la combinación obtiene mejores resultados que GA y NN por si solos.

4. Modelo Matemático

4.1. Modelo de Pritsker (1969)

En el modelo presentado en [19] se toma como parámetro un horizonte de tiempo T que se divide en periodos de tiempo discretos t que se indexan, y se utilizan variables binarias para indicar si una tarea pertenece a un periodo. Esta formulación permite que el problema pueda ser resuelto mediante técnicas de Programación Lineal Mixta. Posteriormente en base al mismo modelo se han presentado variaciones en [9] y en [17].

4.1.1. Variables

Se introduce la variable:

$$y_{(j,t)} = \begin{cases} 1 & \text{si la actividad } j \text{ comienza dentro del periodo } t \\ 0 & \text{si no} \end{cases}$$

donde $j \in V$ y $t = 0, .., T$.

4.1.2. Función Objetivo

$$\text{Min} \sum_{t=0}^{T} t y_{(n+1,t)} \tag{1}$$

4.1.3. Restricciones

- Restricción para que cada actividad solo se inicie una vez dentro del horizonte de tiempo:

$$\sum_{t=0}^{T} y_{(j,t)} = \quad 1 \ , \forall j \in V \tag{2}$$

- Restricción de precedencia entre actividades:

$$\sum_{t=0}^{T} t(y_{(j,t)} - y_{(i,t)}) \geq \quad p_i \ , \forall (i,j) \in A \tag{3}$$

- Restricción de uso de recursos:

$$\sum_{j=0}^{n+1} r_{jk} \sum_{\tau=t-p_j+1}^{t} y_{(j,\tau)} \leq \quad R_k \ , \forall k \in R, \forall t \in \{0, .., T\} \tag{4}$$

- Restricción del tipo de variable:

$$y_{(j,t)} \in \{0, 1\} \ , \forall j \in V, \forall t \in \{0, .., T\} \tag{5}$$

5. Representación

Como representación en este trabajo se utiliza la lista de actividades, mencionada en la sección 3.3.

$$\lambda = \langle j_1, j_2, ..., j_n]$$

En esta representación se mantienen las actividades en una lista ordenada en la secuencia en que van comenzando y siempre mantiene las restricciones de precedencia, es decir una actividad j en la lista siempre cumple con tener todas las predecesoras hacia su izquierda y todas las sucesoras hacia la derecha, para lo cual se hace uso de un operador unario $S(\lambda)$ que genera una solución vecina λ^N seleccionando una actividad j (no ficticia) y realizando un corrimiento circular de actividades en el rango entre su predecesor mas tardío y el sucesor mas temprano. De esta forma el movimiento siempre genera soluciones factibles que respetan la restricción de precedencia (3) del modelo visto en la sección 4.

El mecanismo de selección de j implementado es aleatorio y no utiliza ninguna regla de prioridad. Lo mismo ocurre con la selección de la nueva posición hacia la cual se correrá j.

Asociada a la lista de actividades se mantiene una lista con los tiempos de inicio de cada actividad en la posición, los que son calculados utilizando una heurística basada en SGS serial, indicado en sección 3.2.2, y generando una programación de actividades que cumpla con las restricciones de recursos. El tiempo de inicio de la última actividad en la lista de tiempos corresponde al tiempo total del proyecto, y representa el valor de la función objetivo a minimizar dado en la ecuación (1) del modelo.

Un ejemplo de esta representación y el movimiento se muestran en la figura 3.

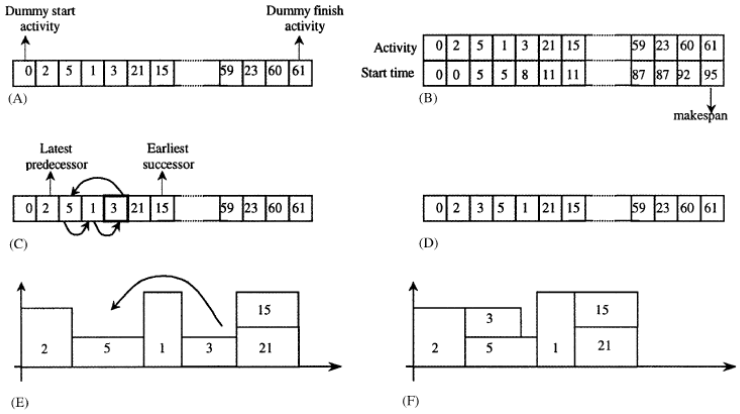

Figura 3: Representación de la solución y movimiento para generar soluciones vecinas.

La lista λ^N y su lista de tiempos asociada son implementadas como vectores que almacenan el identificador de la tarea j y su tiempo de inicio, respectivamente, en una misma posición.

Para generar la programación y los valores en la lista de tiempos a partir de una solución factible λ^N es necesario administrar el uso de los recursos. Esto se logra representando la programación del uso de un recurso k mediante una matriz de tamaño $R_k x T$, donde R_k es la capacidad del recurso y T el horizonte de tiempo máximo indicado en el modelo de la sección 4. En cada casilla se almacena el identificador de la actividad que se encuentra utilizando dicha unidad de su capacidad en una unidad de tiempo t determinado.

La heurística basada en SGS serial debe verificar utilizando estas matrices la disponibilidad para insertar una actividad en la programación de los recursos asociados a la actividad, de manera de cumplir las restricciones indicadas en la ecuación (4.1.3). Además debe asegurar que las asignaciones de una actividad en las matrices sean contiguas al menos en la dimensión del

tiempo, de esta forma manteniendo la restricción dada por la ecuación (2), que restringe a iniciar la actividad solo una vez.

Un ejemplo de la representación para la instancia de ejemplo mostrada en la sección 2 se muestra en la figura 4. Notar que en la implementación real los identificadores de las actividades comienzan por el valor 1, esto de acuerdo al formato de $PSPLIB$.

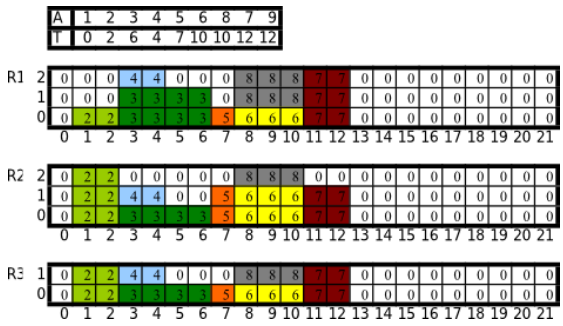

Figura 4: Representación de actividades, tiempos y uso de recursos para la instancia de ejemplo.

6. Descripción del algoritmo

A modo general el algoritmo parte por generar una solución inicial válida respecto a las precedencias. Esto se realiza una única vez utilizando recursión hacia adelante, obteniéndose una solución válida representada por la lista de actividades λ.

A partir de esta solución se genera una solución vecina λ^N con el movimiento $S(\lambda)$ mencionado anteriormente, el cual primero selecciona una actividad j. El método de selección aquí empleado es completamente aleatorio y no utiliza reglas de prioridad. Hay que notar que la utilización de una regla de prioridad podría mejorar la calidad de las soluciones, implementando por ejemplo una selección en base a la menor holgura, es decir, seleccionar primero aquellas actividades mas restringidas por su precedencia.

Luego se calcula el rango comprendido por la actividad predecesora más tardía y la sucesora mas temprana, para seleccionar de manera completamente aleatoria una nueva posición de j, y se realiza un corrimiento cíclico de las actividades.

A partir de la lista λ^N obtenida se utiliza una variante de la heurística SGS serial, el que no genera una lista nueva cada vez, si no que utiliza λ^N como entrada, por lo que solo le basta generar una programación de actividades considerando los recursos. Para esto, por cada actividad j, busca en cada programación del recurso k (matriz de $R_k xT$ que administra el uso del recurso) el tiempo mayor t en que existe disponibilidad para los requerimientos $r_{j,k}$ en la matriz. Una vez determinado t se programa esa actividad en ese periodo t en cada matriz perteneciente a cada recurso, y se actualizan los tiempos de inicio y finalización de la actividad j en la lista de tiempos asociada a λ^N. Al actualizarse el nodo final, se obtiene el valor de duración del proyecto, el cual se busca minimizar.

Esta iteración es implementada en conjunto con la meta heurística Simulated Annealing, comentada en la sección 3.4.2, la cual a diferencia de otras técnicas greedy acepta soluciones peores en función de una probabilidad, lo cual le permite escapar de óptimos locales. La función de probabilidad es controlada por un parámetro T_c denominado temperatura, el cual va decreciendo al ser multiplicado por un parámetro α. El valor inicial de la temperatura debe ser el suficientemente grande para que permita explorar un espacio representativo inicialmente, para luego a medida que disminuye, realizar explotación. El parámetro α también debe ser seleccionado de manera que la temperatura no decaiga muy rápido ni muy lento.

Para este caso puntual se implementó la técnica de Simulated Annealing en un esquema llamado homogéneo, cuyo algoritmo se describe en el listado 1. El algoritmo toma la forma de una cadena de Markov, donde cada cadena es generada en un valor de la temperatura dado. El largo de la cadena es controlado por una variable ml, el cual al superar el parámetro MAX_{ml} hará decrecer la temperatura y generará una nueva cadena. Además si en un número MAX_{rh} no existe mejora se realizará un recalentamiento. Esto es controlado por la variable rh, la cual aumenta cada vez que no existe mejora y vuelve a 0 cuando si la hay. Notar que para controlar en cada iteración se debe comprobar que rh no supere $MAX_{rh} * MAX_{ml}$ iteraciones.

Algorithm 1 Implementación de Simulated Annealing.

```
it ← 0
ml ← 0
rh ← 0
T_c ← T_0
while it < MAX_it do
    it ← it + 1
    ml ← ml + 1
    if ml ≥ MAX_ml then                    ▷ enfriamiento mediante largo de la cadena
        T_c ← T_c * α
        ml ← 0
    end if
    if rh ≥ MAX_rh * MAX_ml then           ▷ recalentamiento mediante cadenas sin mejora
        T_c ← T_0
        rh ← 0
    end if
    S_n ← vecino(S_c)
    Ev_n ← f(S_n)
    if Ev_n < Ev_c then
        S_c ← S_n
        Ev_c ← Ev_n
        rh ← 0
    else
        rh ← rh + 1
        if random(0,1) < P(Ev_c, Ev_n, T_c) then
            S_c ← S_n
            Ev_c ← Ev_n
            rh ← 0
        end if
    end if
    if Ev_c < Ev_best then
        S_best ← S_c
        Ev_best ← Ev_c
    end if
end while
```

7. Experimentos

Para la experimentación con el algoritmo se utilizaron las instancias de PSPLIB [13], las cuales están clasificadas en grupos de modo único y multimodo, este último descartándose ya que la implementación actual solo contempla el modo simple. El conjunto de instancias

se encuentra separado en 4 grandes grupos según la cantidad de actividades, en 30, 60, 90 y 120 actividades, sin contar la inicial y la final. Además se cuenta con una lista de las mejores soluciones encontradas, y de óptimos globales en el caso de instancias de 30 actividades. Con estos valores es posible comparar el algoritmo con los mejores resultados.

Primero se realizó una exploración variando los siguientes parámetros, realizando 10 ejecuciones con los mismos parámetros y revisando el promedio, desviación estándar y el mínimo encontrado, esto para cada una de los cuatro grupos de N, con 30, 60, 90 y 120 actividades.

- MAX_{it}: El número total de iteraciones, con valores en el conjunto {1000, 5000, 10000, 50000, 100000}. Es natural pensar que a mayor cantidad de iteraciones se encontrarán mejores resultados.
- T_0: Temperatura inicial. Para el caso de la temperatura se probaron valores en el conjunto {500, 100, 90, 80, 60}. Para este parámetro en algunas implementaciones de Simulated Annealing se recomendaba utilizar un valor extremadamente alto, dado que la disminución se realizaba en cada iteración, pero en el esquema homogéneo, no ocurre así.
- α: Factor de enfriamiento, con valores en el conjunto {0.999, 0.99, 0.98, 0.97, 0.95, 0.9, 0.8, 0.6}.

Esto para ver como inciden estos parámetros por si solos en la ejecución del algoritmo y la búsqueda de soluciones, sin modificar los otros parámetros, que son:

- MAX_{ml}: Largo de la cadena de Markov, es decir, el número máximo de soluciones generadas por cada temperatura.
- MAX_{rh}: La cantidad máxima de cadenas a generar sin que se realice un recalentamiento.

Primero se comenzó utilizando la instancia j601_1 para la de 60 actividades, pero al probar el número de iteraciones se notó que se llegaba rápidamente al óptimo y la variación en las iteraciones no entregaba buena información, por lo que se decidió cambiar por la instancia j601_2.

Luego de revisar estos parámetros se decidió experimentar con los otros dos parámetros, manteniendo constante la cantidad de iteraciones y para una misma instancia observar como se aproximaba el valor de la función objetivo versus las iteraciones del algoritmo. Primero se revisó una ejecución de una instancia de 30 actividades, para luego ir variando los parámetros y observar los puntos de recalentamiento. Luego se realiza lo mismo para una instancia de 120 actividades.

8. Resultados

Con respecto al diseño e implementación del algoritmo, se logran incorporar los mecanismos indicados en el estado del arte, esto es una técnica meta heurística como Simulated Annealing, que utiliza correctamente los parámetros para controlar los estados de energía. Además se implementa una heurística de generación de scheduling serial (SGS serial). Tanto los movimientos que generan vecindades como la heurística, generan soluciones posibles y que cumplen todas las restricciones, por lo que no es necesario incluir penalizaciones en la función objetivo.

La implementación además resulta bastante rápida y eficiente utilizando entre 800 KB y 1 MB para los problemas más grandes de 120 actividades.

Para la primera parte, variación del máximo de iteraciones, era de esperar que se notara que a mayor cantidad de iteraciones se obtuvieran mejores resultados, lo que se puede comprobar observando los promedios obtenidos en los datos de la tabla en la figura 6.

En la variación de temperatura inicial se obtuvieron los datos de la tabla en la figura 7. No se observa alguna relación con los promedios y no es posible obtener información útil de las

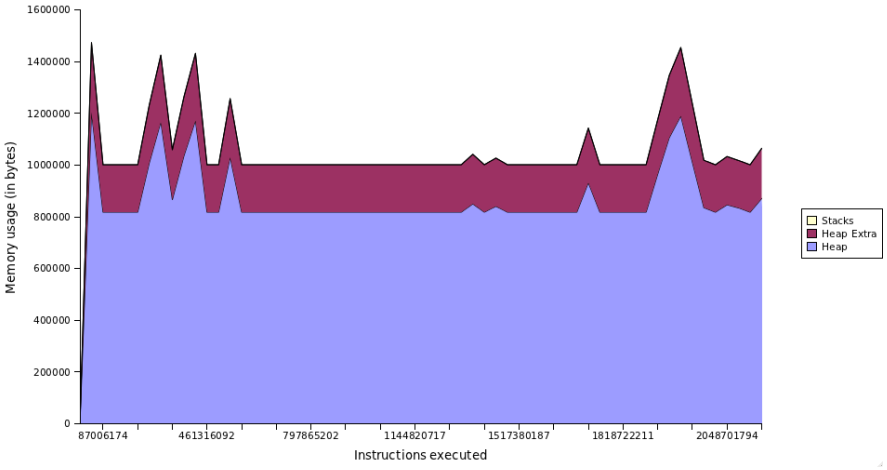

Figura 5: Utilización de la memoria RAM durante la ejecución sobre una implementación de 120 actividades

	IT	AVG	SD	MIN
301_1	1000	48.1	0.875595	46
	5000	47.4	0.843274	46
	10000	47.5	0.527046	47
	50000	46.2	0.421637	46
	100000	46.1	0.316228	46
601_2	1000	76.7	1.337494	75
	5000	74.1	2.024846	72
	10000	72.8	1.619328	70
	50000	71.4	1.074968	70
	100000	71	0.816497	70
901_1	1000	83.5	1.433721	82
	5000	82.2	1.135292	80
	10000	80.8	0.788811	80
	50000	79.2	1.316561	77
1201_1	1000	119.3	1.159502	117
	5000	117.8	1.686548	116
	10000	117.3	1.418136	115
	50000	115.6	1.955050	113

Figura 6: Variación del número máximo de iteraciones MAX_{it}

desviaciones estándar al menos sin conocer como varía la solución a lo largo de las iteraciones, la cual debería mostrar una mayor variación hacia el comienzo de la ejecución del algoritmo a mayor temperatura inicial.

La variación del parámetro α tampoco muestra grandes diferencias en los valores y no es posible establecer alguna tendencia con los promedios o las desviaciones estándar. Es posible notar, revisando las tres tablas anteriores que siempre la instancia de 60 actividades presenta las desviaciones estándar mayores, y la instancia de 30 las menores.

Luego para visualizar los cambios en las soluciones se grafica la ejecución del algoritmo sobre una instancia de 30 actividades, mostrando las evaluaciones de la función objetivo actual (Ev_c) y la mejor (Ev_best), a lo largo de las iteraciones. Se pueden ver los resultados en la figura 9. Además se grafica la variación de la temperatura en el otro eje, y se observan fácilmente dos recalentamientos, además de como la disminución de temperatura va afectando la aceptación de empeoramientos y como Simulated Annealing parte por explorar y poco a poco comienza a intensificar la búsqueda.

Se realiza el mismo gráfico sobre una instancia de 120 actividades, obteniéndose el gráfico de

	T0	AVG	SD	MIN
301_1	500	47	0.94280904158206	46
	100	47.3	0.82327260234856	46
	90	47.1	0.87559503577091	46
	80	47.5	0.70710678118655	46
	60	47.4	0.9660917830793	46
601_2	500	73.5	1.90029237516523	71
	100	73.6	1.7763883459299	72
	90	73.2	1.93218356615859	71
	80	73.5	1.43372087784044	72
	60	73.7	1.56702123647242	72
901_1	500	82.3	1.05934990547138	81
	100	81.9	1.10050493461461	80
	90	81.6	1.07496769977314	80
	80	81.2	1.13529242439509	79
	60	81.4	0.9660917830793	80
1201_1	500	117.3	1.41813649241218	115
	100	117.4	1.26491106406735	115
	90	116.8	0.91893658347268	115
	80	117.5	0.8498365855988	116
	60	117.7	1.05934990547138	116

Figura 7: Variación de la temperatura inicial T_0

la figura 10. Se observa que para este caso no alcanza a ocurrir un recalentamiento, por lo que la búsqueda se intensifica y se queda en una zona. Esto ocurre con los parámetros $MAX_{ml} = 200$ y un factor $MAX_{rh} = 100$, por lo que para lograr un recalentamiento se prueba modificándolos por $MAX_{ml} = 100$ y $MAX_{rh} = 50$, obteniéndose el resultado de la figura

9. Conclusiones

Como conclusiones del estudio, se ha explorado el problema RCPSP presentando sus características principales y sus variantes. Se concluye que las técnicas de búsqueda completa se pueden utilizar para resolver solo para instancias pequeñas y que para instancias mayores se recomienda el uso de heurísticas. Existen variaciones por cada caso que intente modelar mejor el problema real, pero estas por lo general introducen un mayor número de restricciones, haciendo que el problema se vuelva más complejo de resolver. Por ejemplo si se utilizan variaciones multimodo el problema MRCPSP debe considerar K restricciones por cada actividad j, por lo que el problema crece exponencialmente por cada incremento en el número de actividades o de modos. En base al Estado del Arte se pude concluir que existe una tendencia en utilizar PSPLIB [13] como una base en común de instancias para comparar los algoritmos lo que permite una mayor objetividad. Además en cuanto a los métodos de resolución se puede concluir que la tendencia va hacia el uso de meta-heurísticas para búsquedas globales utilizando las heurísticas de generación de secuencias, y luego en base a alguna técnica de optimización local (como redes neuronales o lógica difusa) se busca mejorar la solución, la cual es luego utilizada en el algoritmo de búsqueda global.

Como conclusiones de la experimentación se puede observar que la implementación logra comportarse acorde a lo que se esperaba y los parámetros son capaces de modificar la exploración y la intensificación. Además las evaluaciones de las soluciones se aproximan a la lista de soluciones dada por PSPLIB. Se puedo observar de los gráficos que ocurre una rápida transición en que la variación de la solución es alta y se aceptan muchas soluciones peores a un estado en que bajan considerablemente. Esto puede deberse a la generación de una solución en que las posibilidades de movimientos que mejoren la función objetivo pasen por mover una actividad muy restringida siendo necesariamente empeorar la solución para alcanzar mejores posibilidades de movimientos para las demás actividades. Para mejorar esto es posible incorporar a la selección de actividades alguna regla de prioridad.

	ALPHA	AVG	SD	MIN
301_1	0.999	47.4	0.51639778	47
	0.99	47.3	0.67494856	46
	0.98	47.1	0.87559504	46
	0.97	47.5	0.84983659	46
	0.95	46.9	0.73786479	46
	0.9	47.4	0.69920590	46
	0.8	47.5	0.70710678	46
	0.6	47.4	0.69920590	46
601_2	0.999	73	2.44948974	69
	0.99	73.1	1.59513148	72
	0.98	73	1.63299316	72
	0.97	73	1.49071198	72
	0.95	72.6	1.0749677	72
	0.9	72.7	1.15950181	71
	0.8	73	1.94365063	71
	0.6	72.8	2.25092574	69
901_1	0.999	81.8	1.31656118	80
	0.99	81.7	0.82327260	81
	0.98	81.5	1.35400640	79
	0.97	81.3	1.25166556	79
	0.95	79.7	1.41813649	78
	0.9	80.1	1.52388393	78
	0.8	82	1.88561808	79
	0.6	81.4	1.71269768	79
1201_1	0.999	116.7	1.88856206	112
	0.99	117.2	1.54919334	115
	0.98	117.1	0.87559504	116
	0.97	117.1	0.99442893	116
	0.95	116	1.63299316	113
	0.9	115.3	1.15950181	114
	0.8	116.8	1.39841180	114
	0.6	115.3	1.56702124	113

Figura 8: Variación de la ponderación de temperatura α

Aún con las posibles mejoras que se le puedan realizar a la implementación, las soluciones encontradas son bastante buenas a un costo computacional modesto. Como comparación, la resolución de la instancia de 120 actividades para 50000 iteraciones es resuelta en al rededor de 50 segundos.

10. Bibliografía

[1] Anurag Agarwal, Selcuk Colak, and Selcuk Erenguc. A neurogenetic approach for the resource-constrained project scheduling problem. *Computers & Operations Research*, 38(1):44 – 50, 2011. Project Management and Scheduling.

[2] Christian Artigues, Philippe Michelon, and Stéphane Reusser. Insertion techniques for static and dynamic resource constrained project scheduling. *European Journal of Operational Research*, 149:249–267, 2003.

[3] T. Baar, P. Brucker, and S. Knust. Tabu search algorithms and lower bounds for the resource-constrained project scheduling problem. In *MIC-97: meta-heuristics international conference*, pages 1–18, 1999.

[4] J. Blazewicz, J.K. Lenstra, and A.H.G.Rinnooy Kan. Scheduling subject to resource constraints: classification and complexity. *Discrete Applied Mathematics*, 5(1):11 – 24, 1983.

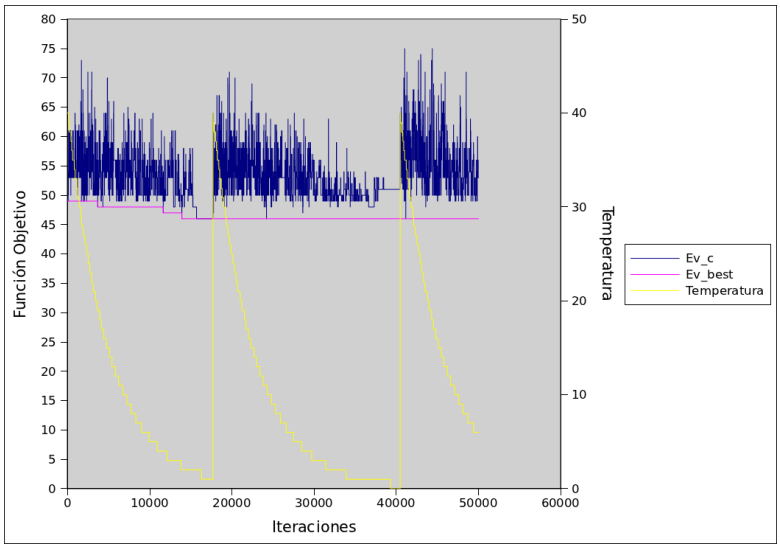

Figura 9: Ejecución de Simulated Annealing sobre instancia de 30 actividades

[5] K. Bouleimen and H. Lecocq. A new efficient simulated annealing algorithm for the resource-constrained project scheduling problem and its multiple mode version. *European Journal of Operational Research*, 149(2):268 – 281, 2003. Sequencing and Scheduling.

[6] P. Brucker and S. Knust. A linear programming and constraint propagation-based lower bound for the rcpsp. *European Journal of Operational Research*, 127(2):355–362, 2000.

[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*, 107(2):272–288, 1998.

[8] Peter Brucker, Andreas Drexl, Rolf Möhring, Klaus Neumann, and Erwin Pesch. Resource-constrained project scheduling: Notation, classification, models, and methods. *European Journal of Operational Research*, 112(1):3 – 41, 1999.

[9] N. Christofides, R. Alvarez-Valdés, and J.M. Tamarit. Project scheduling with resource constraints: A branch and bound approach. *European Journal of Operational Research*, 29(3):262–273, 1987.

[10] M. Dorigo, G. D. Di Caro, and L. Gambardella. Ant Colony Optimization: A New Meta-Heuristic. In Peter J. Angeline, Zbyszek Michalewicz, Marc Schoenauer, Xin Yao, and Ali Zalzala, editors, *Proceedings of the Congress on Evolutionary Computation*, volume 2, pages 1470–1477, Mayflower Hotel, Washington D.C., USA, Jun-Sep 1999. IEEE Press.

[11] S. Hartmann and A. Drexl. Project scheduling with multiple modes: A comparison of exact algorithms. *Networks*, 32(4):283–297, 1998.

[12] W.S. Herroelen. Resource-constrained project scheduling–the state of the art. *Operational Research Quarterly*, pages 261–275, 1972.

[13] R. Kolisch and A. Sprecher. Psplib-a project scheduling problem library: Or software-orsep operations research software exchange program. *European Journal of Operational Research*, 96(1):205–216, 1997.

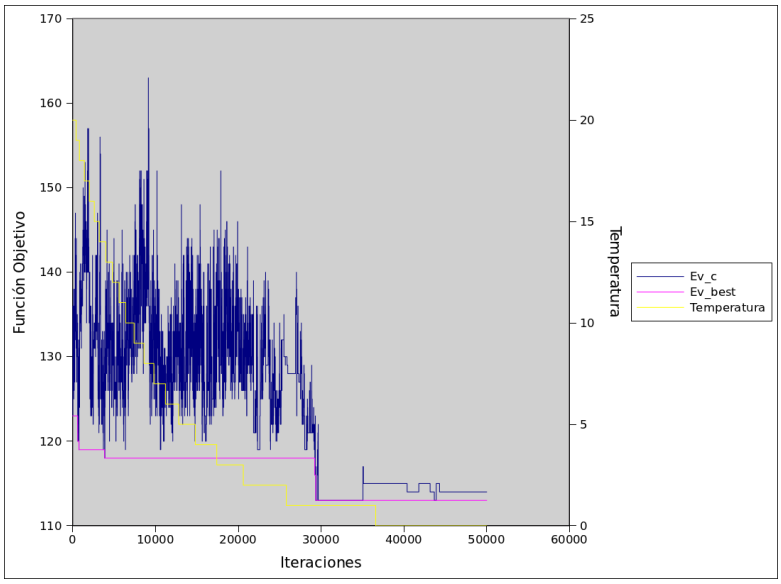

Figura 10: Ejecución de Simulated Annealing sobre instancia de 120 actividades

[14] Rainer Kolisch and Sönke Hartmann. Experimental investigation of heuristics for resource-constrained project scheduling: An update. *European Journal of Operational Research*, 174(1):23 – 37, 2006.

[15] A. Lova, P. Tormos, and F. Barber Sanchís. Multi-mode resource constrained project scheduling: scheduling schemes, priority rules and mode selection rules. *Inteligencia artificial: Revista Iberoamericana de Inteligencia Artificial*, 10(30):69–86, 2006.

[16] D. Merkle, M. Middendorf, and H. Schmeck. Ant colony optimization for resource-constrained project scheduling. *Evolutionary Computation, IEEE Transactions on*, 6(4):333 – 346, aug 2002.

[17] R.H. Möhring, A.S. Schulz, F. Stork, and M. Uetz. Solving project scheduling problems by minimum cut computations. *Management Science*, pages 330–350, 2003.

[18] J.H. Patterson, F. Brian Talbot, R. Slowinski, and J. Weglarz. Computational experience with a backtracking algorithm for solving a general class of precedence and resource-constrained scheduling problems. *European Journal of Operational Research*, 49(1):68–79, 1990.

[19] A. Alan B. Pritsker, Lawrence J. Waiters, and Philip M. Wolfe. Multiproject scheduling with limited resources: A zero-one programming approach. *Management Science*, 16(1):93–108, 1969.

[20] Vicente Valls, Francisco Ballestin, and Sacramento Quintanilla. Justification and rcpsp: A technique that pays. *European Journal of Operational Research*, 165(2):375 – 386, 2005. Project Management and Scheduling.

[21] J.D. Wiest. Some properties of schedules for large projects with limited resources. *Operations Research*, pages 395–418, 1964.

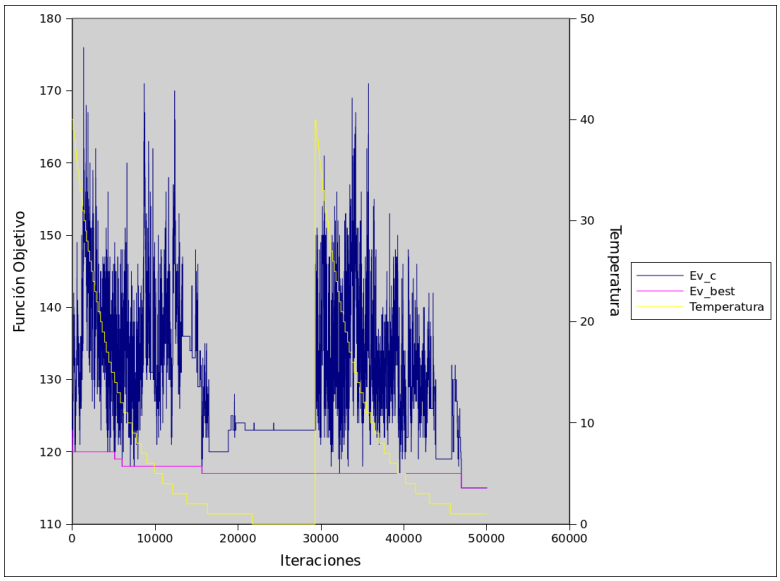

Figura 11: Ejecución de Simulated Annealing sobre instancia de 120 actividades con recalentamiento