TÍTULO: "Algoritmo de optimización económica de carteras de proyectos: Aplicación de subastas combinatorias"

AUTOR: Carolina Rodríguez Pulgar

DIRECTOR: Adolfo López Paredes

DEPARTAMENTO: Organización de empresas y C.I.M

CO-DIRECTOR: Félix Antonio Villafáñez Cardeñoso

TITULACIÓN: Ingeniería Industrial (Plan 210)

CONVOCATORIA: Julio, 2013

RESUMEN:

Réplica de un algoritmo para optimización y selección de carteras de proyectos. Se trata de una aplicación desarrollada en Netlogo en la que se han incorporado mejoras de carácter económico en la valoración de proyectos y nuevos criterios de decisión en el algoritmo, que proponen nuevas soluciones.

Se completa con un análisis de resultados al ejecutar varias carteras (por el método original y por el nuevo propuesto) cambiando diferentes variables de la programación.

PALABRAS CLAVES:

Project Scheduling, Project Management, Carteras, Netlogo, Optimización, Proyectos.

Universidad de Valladolid

ESCUELA DE INGENIERÍAS INDUSTRIALES

DEPARTAMENTO DE ORGANIZACIÓN DE EMPRESAS Y C.I.M.

PROYECTO FIN DE CARRERA

ALGORITMO DE OPTIMIZACIÓN ECONÓMICA DE CARTERAS DE PROYECTOS: APLICACIÓN DE SUBASTAS COMBINATORIAS

Dirigido por Dr. Adolfo López Paredes

Co-dirigido por D. Félix Antonio Villafáñez Cardeñoso

Realizado por Dña. Carolina Rodríguez Pulgar

Valladolid, Julio 2013

INDICE

I. INTRODUCCIÓN

El *Project Scheduling* o Programación de proyectos, es una rama de investigación relativamente moderna, pero que actualmente está comenzando a tener una presencia muy importante en el mundo científico en general, y también en el ámbito ingenieril. En los últimos años, el número de publicaciones en relación al desarrollo e investigación en la teoría de programación de proyectos se ha visto incrementado exponencialmente, y los conocimientos en torno al llamado Project Management están comenzando a ser muy valorados por las organizaciones.

Pues bien, ¿por qué esta relevancia?, ¿qué convierte al Project Scheduling en una tendencia de estudio tan importante a día de hoy? En esta introducción pretendemos justificar esta realidad y ver qué aportación pretende dar este PFC y dónde podemos ubicarlo dentro de los numerosos bloques de investigación que podemos encontrar.

De forma cada vez más habitual, vemos como las empresas tienden a seguir una organización basada en proyectos, es decir, frente a la estructura más tradicional donde del organigrama general se desplegaban todos los departamentos propios de una empresa de forma independiente (administración, marketing, ventas, logística, I+D…), ahora se presenta una estructura dividida en proyectos, donde las fuerzas se ven divididas en cada uno de ellos. Aún no está del todo justificado, pero si que se está viendo como empresas con este tipo de organización son las que terminan siendo más exitosas y sobreviven en situaciones más complicadas.

Por todo ello, la situación que comenzamos a ver en cada vez un mayor número de empresas es la de tener la oportunidad de acometer varios proyectos, los cuáles consumirán un conjunto de recursos, tendrán una serie de actividades definidas, un horizonte temporal en que verse realizados, unas exigencias de financiación… Todas estas variables, junto con las restricciones que nuestra organización pueda llegar a tener, hacen que optimizar este problema sea habitualmente complicado de resolver.

Al fin y al cabo, una simplificación de nuestro objetivo sería saber, de entre todos los proyectos que conformen nuestra cartera (según el criterio que decidamos, y ahí entrará en juego el Project Manager en cuestión), cuáles nos merece la pena ejecutar, cuáles es mejor desestimarlos, y entre aquellos candidatos afortunados, qué planificación tener para ellos. Y junto con todo ello, asegurarnos que en todo momento, la organización podrá cubrir el consumo de recursos, así podremos hablar de una buena planificación. Si, además de una solución factible, deseamos tener la óptima (la que implique menos tiempo, tenga un menor consumo medio o sea la más rentable), se complica aún más su obtención.

Por todo ello, se pretende obtener formas de abordar este problema, algoritmos, heurísticas que lo resuelvan (o al menos den una solución próxima a la óptima), criterios para saber cómo afrontar cada cartera según sus características… Vemos como es un área de investigación viva y que aún tiene mucho por explorar.

Lo que suscita el presente PFC es la publicación en 2010 de un artículo titulado *"Combinatorial auction algorithm for project portfolio selection and scheduling to maximize the net present value"* (Shou & Huang), el cuál propone un método de selección y optimización de carteras de proyectos. En este artículo se presenta el aparato matemático que hay detrás de la alternativa que proponen y la solución que obtienen al ejecutar el algoritmo en una cartera ejemplo.

En relación al tipo de planteamiento en este artículo, según la bibliografía estaríamos hablando de un PPSSP (Project Portfolio Selection and Scheduling Problem), concepto que desarrollaremos en más profundidad en este informe, pero que tiene como idea general el seleccionar de una cartera una subcartera de proyectos a ejecutar, y de ahí obtener la programación óptima y factible con las actividades de todos ellos.

El primer objetivo de este PFC era la réplica de este algoritmo con el programa de simulación de agentes Netlogo, por sus grandes ventajas en relación a las salidas visuales. Una vez tuviéramos la programación, podríamos hacer el análisis o bondad del algoritmo ejecutando varias carteras y valorando los resultados obtenidos.

Por otra parte, el algoritmo original tenía puntos y propuestas susceptibles de mejorar, por ello se han incorporado algunas mejoras (las cuáles también analizaremos). Las dos aportaciones principales han sido:

- En relación a la valoración de proyectos: el tratamiento de los flujos de caja en el tiempo no se consideraba con la tasa de descuento habitual, k, de tal forma que su descripción del VAN no es la reconocida popularmente. Su planteamiento era disminuir en un porcentaje el valor máximo del proyecto conforme se aleja de su tiempo mínimo de ejecución.

- En relación a la heurística de asignación de prioridades: una vez que tenemos toda una ristra de actividades para ser ordenadas en la programación, debemos definir un criterio de prioridad. Encontramos de muy diversos tipos, y esta decisión condiciona directamente la solución que se obtenga. La opción que planteamos con respecto a la del artículo, incorpora el valor máximo del proyecto, de ahí que una actividad tendrá mayor prioridad si pertenece a un proyecto que reporta mayor beneficio.

Cabe comentar como el ejemplo resuelto en la publicación tiene más bien un carácter didáctico para entender bien el algoritmo que proponen. Aborda proyectos bastante sencillos (pocas actividades, reglas de precedencia simples) y sólo pone en un juego un único recurso, de tal forma,

que las necesidades de los proyectos en la cartera se reducen a unidades de este recurso, y por tanto la restricción global para toda la cartera también es sencilla. Como vemos, vuelve a ser planteamiento bastante ideal, ya que es lógico que un proyecto en la realidad necesite más de un recurso para verse acometido. Por ello, la programación propuesta está diseñada para que funcione hasta con cuatro recursos, lo que, como ya veremos, no complica conceptualmente el problema, pero sí que influirá mucho en la posible solución final.

Por último, este PFC pretende demostrar cuáles son las debilidades que, incluso tras nuestras incorporaciones, somos conscientes que sigue sufriendo el algoritmo, y cuáles serían las próximas vías de investigación que podrían continuar futuros compañeros.

II. PROJECT SCHEDULING. CONCEPTOS PREVIOS

Una buena práctica antes de poder abordar cualquier tema en profundidad, y más en el ámbito de la investigación, sea cual sea el campo en cuestión, es comenzar interesándose por qué es lo que se ha conseguido hasta el momento en ese terreno. Este historial de sucesos vinculados al estudio sobre un tema es conocido como el *estado del arte*.

Las razones son más que evidentes; saber desde qué punto comienza la búsqueda de nuestra aportación, evitar esfuerzos innecesarios o poder documentar debidamente nuestros informes con el apoyo bibliográfico oportuno. Si la investigación puede ser concebida como un trabajo en equipo en varios aspectos, parece básico conocer las tendencias de tus compañeros.

Por todo ello, este proyecto fin de carrera parte del resumen de los conceptos más utilizados en el ámbito de la programación de proyectos y de las vertientes más representativas y actuales en el Project Scheduling y la optimización de carteras.

Estamos hablando de una rama de estudio muy joven con respecto a otras, de hecho, las primeras propuestas del mundo científico en esta vía se remontan a hace unos 15 años aproximadamente. Esta característica hace del Project Scheduling una opción muy atractiva para miles de investigadores, puesto que aún queda mucho por caminar.

Este capítulo será el perfecto pilar para entender el interés y la aportación que este documento pretende dar a toda la compleja teoría ya desarrollada en estos últimos años.

II.a. ¿Qué es un proyecto? La relevancia de la programación de proyectos

En mayor o menor medida, todas las empresas conviven en una situación de competitividad continua. Además, el concepto de "aldea global" para nuestro mundo, cada vez es más evidente, lo que nos exige ser los mejores a gran escala.

Esta es una de las razones por las que debemos tener presente ofrecer productos/servicios de calidad, en el menor tiempo posible y consumiendo el menor número de recursos (piezas, máquinas, personas…). Al final, el objetivo de una empresa termina siendo gestionar proyectos (propios o ajenos) de la forma más eficiente posible, sometidos a unas restricciones. De ahí, el más que justificado interés por optimizar este proceso y conseguir solventar el problema que en gran parte condicionará el éxito de nuestra empresa. La teoría de gestión de proyectos es, por tanto, algo vivo, en desarrollo y en constante evolución.

El término "proyecto" tiene muchos matices que abordar, por ello no hay una definición sumamente aceptada por todos. Una posible opción clara y completa, sería:

> *Un proyecto es un único proceso, compuesto por un conjunto de actividades coordinadas y controladas con una fecha de inicio y fin, para alcanzar un objetivo según los requerimientos de restricciones de tiempo, coste y recursos.* (Demeulemeester, E., Herroelen, 2002)

También es importante añadir a esta definición algunas consideraciones:

- La organización del proyecto es temporal y se establece a lo largo de la vida del proyecto.
- En muchas ocasiones un proyecto puede formar parte de una estructura mayor de proyectos (cartera de proyectos).
- Los objetivos del proyecto deben ir alcanzándose progresivamente en el curso de éste.
- Las relaciones entre las actividades del proyecto pueden llegar a ser complejas.

A pesar de la diversidad de proyectos que podemos encontrar, podemos encontrar parámetros comunes en todos ellos:

- **Objetivo:** meta del proyecto que habitualmente es definida en términos de costes, calidad del producto o bien, en relación a una fecha límite.

- **Unicidad:** incluso proyectos que parecen idénticos, ya parten de tener contextos temporales diferentes, luego cada uno debe ser abordado de forma exclusiva. Este detalle no resta el beneficio que la experiencia en el proyecto previo nos puede aportar.
  Es esta característica la que muchos autores consideran como más relevante.

- **Complejidad:** las propias relaciones entre las actividades o las restricciones, pueden llegar a ser muy complejas.

- **Inciertos:** al ser programadas antes de su ejecución, hay un factor riesgo intrínseco a todos los proyectos.

- **Ciclo de vida:** a lo largo del proyecto hay diferentes fases, desde la del diseño cuando se nos hace la petición por parte del cliente, hasta la definición del proyecto aclarando cómo será toda la ejecución. En la planificación es importante marcar todas las actividades que se van a tener que llevar a cabo (orden, duraciones, estimación de recursos involucrados). Una vez hemos alcanzado la descripción detallada de todas las actividades, entra ya en juego la programación del proyecto (*Scheduling*) que supone el reordenamiento de las actividades, respetando precedencias y con un uso de recursos factible y eficiente. Además, puede haber restricciones limitantes en relación al coste máximo del proyecto, o al tiempo

máximo de ejecución, que también podrían considerarse en la programación. Durante la ejecución de éste, debe haber un proceso de control y monitorización con intención de poder tomar acciones correctivas si fueran necesarias. En el siguiente punto abordaremos más en profundidad cada fase.

La gestión de proyectos está involucrada a todo el ciclo de vida del proyecto: planificación, programación y control de las actividades, para concluir el proyecto con unos objetivos temporales y en costes, usando siempre los recursos de la forma más eficiente.

Si bien, un proyecto es abordado con éxito cuando se finaliza en tiempo, bajo el presupuesto inicial y cumpliendo las especificaciones originales, es bastante habitual encontrar casos en la vida real donde alguna o varias de estas condiciones no son respetadas. Por tanto, de un buen gestor de proyectos se espera resultados exitosos, lo que supone un gran conocimiento sobre la programación de proyectos y sobre el trabajo bajo condiciones de riesgo.

La International Project Management Association (IPMA) otorga certificados individuales (además de a organizaciones, consultorías…) que verifican la competencia en gestión de proyectos de sus portadores.

II.b. Representación gráfica de un proyecto

Como hemos anticipado, la fase de programación del proyecto es principal, ya que condiciona directamente la resolución y la buena ejecución de este. Es por ello que, gran parte de la investigación y estudio en la gestión de proyectos, pretende encontrar métodos, algoritmos, heurísticas, cada vez más eficaces y eficientes.

En este apartado se pretende presentar algunas de las herramientas más importantes usadas en la programación de proyectos a día de hoy, y más conceptos utilizados habitualmente en la bibliografía.

De forma simplificada, un proyecto está compuesto por un conjunto de **hitos** (puntos relevantes a lo largo de la vida del proyecto) y de **actividades** (o tareas) que son secuenciadas según unas reglas de precedencia. Cada una de estas actividades tiene su duración asociada y por lo general, un consumo de recursos (en el caso de que no consuma ningún recurso, hablaremos de unas actividades particulares conocidas como *dummies*). Los recursos pueden ser de todo tipo: financieros, humanos, equipamientos, materiales…

Es fácil entender como las precedencias entre actividades son restricciones básicas en un proyecto, ya que en ocasiones puede no ser viable el desarrollo de una actividad sin haberse llevado

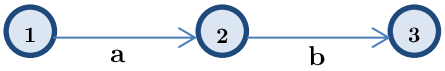
a cabo una o varias antes. Las relaciones más habituales son las de final – inicio, es decir, no hay espacio temporal desde que una precedente termina y comienza su sucesora. En todo caso, no habría ningún problema en valorar relaciones de cualquier otro tipo (inicios o finales simultáneos, separadas por tiempo ocioso…).

La forma más común de representación de un proyecto es mediante un grafo, donde de forma visual quedan presentes todas las relaciones de precedencia entre las actividades. Un grafo G, está compuesto por V vértices (nodos) y E aristas (arcos), de tal forma que habitualmente usaremos la notación $G=(V,E)$.

Hay dos formas principales de representación de grafos: *Activity-on-arc* (AoA) o *Activity-on-node* (AoN).

Activity-on-arc (AoA):

Cada actividad queda representada por un arco y cada evento por un nodo. Por tanto, para representar que no hay retraso entre dos actividades a y b, el nodo final de la actividad a, debe coincidir con el nodo inicial de b, como indica la figura. En ese caso, la actividad a es predecesora inmediata de b.

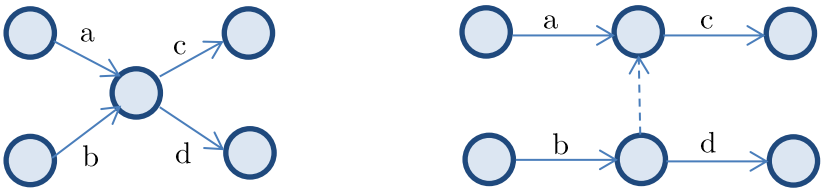


Ejemplo grafo de actividades sucesivas en AoA

Los nodos quedan numerados de tal forma que no haya eventos duplicados y habitualmente, el sentido del arco va de un nodo de menor a mayor numeración. Es común representar las conexiones entre los nodos a través de una matriz de adyacencia. Se trata de una matriz $n \times n$ siendo n el número total de nodos. Sólo admite valores binarios, y en el caso de que aparezca un 1 nos indica que efectivamente hay una actividad entre esos dos nodos. Herramientas de este tipo son, por ejemplo, usadas en la representación de redes de transporte.

Además, siendo consecuentes con la programación, una actividad sólo tiene un origen y un final, luego entre dos nodos cualesquiera, sólo estará presente un arco. En ocasiones, es necesario recurrir a actividades ficticias (no consumen ni tiempo ni recursos y no modifican la lógica de la red) para la correcta representación de las redes. Algunas razones pueden ser:

- *Asegurar la exclusividad de todas las actividades (en el caso de que varias actividades se desarrollaran simultáneamente, no hay dos con el mismo origen y final).*
- *Conseguir un único nodo origen (que no tiene precedentes) y nodo final (que no tiene sucesores).*
- *Representar las relaciones de precedencia correctamente, de tal forma que no se puedan hacer interpretaciones equívocas al ver el grafo. Un ejemplo sería el siguiente:*



Como vemos, en la primera figura se podría interpretar que para realizar la actividad d tendríamos como antecedentes las actividades a y b. En la segunda figura, al introducir la actividad ficticia, eliminamos la precedencia de a.

Muchos algoritmos (Dimsdale (1963); Fisher et al. (1968); Hayes (1969)…) han sido desarrollados para minimizar el número de actividades ficticias en una red AoA. Este problema es de carácter NP-hard. Tratamos brevemente este concepto:

¿Qué es un problema NP-hard?

Este atributo está vinculado a la complejidad computacional del problema. En la literatura podemos encontrar tanto algoritmos de programación que pueden abordar miles de actividades, como otros que sólo permiten un pequeño número.

Los problemas computacionales pueden verse como una función f con sus entradas x y sus salidas $f(x)$. La complejidad de éstos está vinculada al tiempo computacional que requiere la resolución de dicha función, así encontramos problemas sencillos (easy) o complejos (hard). En primera aproximación, podemos relacionarlo al número de iteraciones que debe realizar el algoritmo para cada entrada x.

Hay algoritmos cuyo tiempo de ejecución está limitado por una función polinomial de n, que es la longitud del input x en una instancia del problema. Para la mayoría de problemas, lamentablemente, no se encuentran algoritmos de tiempo polinomial, si no exponencial.

Por otra parte, todo problema de programación puede ser formulado como uno de decisión, es decir:

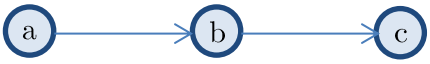
"Encuentra la programación factible más rápida en el tiempo"	*"¿Hay una programación factible que se complete en un determinado plazo?"*
Problema de optimización	***Problema de decisión***

Como vemos, el planteamiento como problema de decisión admite una contestación de Sí/No. Si un problema de decisión es complejo, también lo será el de optimización asociado. En estos problemas de decisión, hablaremos de un problema de clase NP cuando no se conocen

algoritmos de resolución en tiempo polinomial, pero sólo la contestación Sí/No a la pregunta oportuna, sí se puede confirmar en tiempo polinomial.

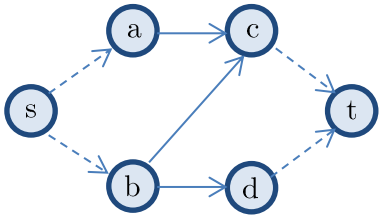
Activity-on-node (AoN):

Cada actividad queda representada por un nodo y los arcos simbolizan las reglas de precedencia requeridas entre las actividades.



Ejemplo grafo de actividades sucesivas en AoN

En este caso, las actividades ficticias (nodos ficticios en este caso), sólo aparecen si es necesario tener en la red un único nodo inicial (s) y final (t). Un ejemplo sería la siguiente figura, en la que tendríamos dos actividades simultáneas al inicio y otras dos concluyen al final del proyecto.



Uso de nodos ficticios como recurso

La opción AoN permite representar otro tipo de relaciones de precedencia diferentes a las más habituales de final – inicio.

II.c. Las relaciones de precedencia

De forma sencilla, describimos una relación de precedencia como la restricción temporal que existe entre una actividad y otra sucesora. Podemos distinguir cuatro tipos de relaciones:

- **SS (start – start):** un cierto tiempo debe transcurrir entre los comienzos de dos actividades, es decir, parte de una actividad debe haber concluido para que empiece la otra.

- **SF (start – finish):** son menos usadas. El tiempo se mide entre el comienzo de una actividad y el final de otra.

- **FS (finish – start):** las que ya habíamos mencionado. Para comenzar la siguiente debe haber finalizado la anterior, algo muy habitual en proyectos.

- **FF (finish – finish):** en este caso, el final de una actividad está condicionado al final de otra. Por ejemplo, si pensáramos en pintar el techo de una habitación, el final de esta actividad estaría condicionado a cuándo terminen de colocar las molduras de escayola.

La notación de todas ellas, sería de la forma $SS_{ij}^{\min}(x), SF_{ij}^{\min}(x), FS_{ij}^{\min}(x), FF_{ij}^{\min}(x)$ donde indicaríamos el número de unidades de tiempo x exigidas entre el inicio (final) – inicio (final) de las actividades i y j. En este caso, hemos tomado como ejemplo, que la restricción nos hable del mínimo tiempo que al menos debe haber entre los dos sucesos, pero podrían indicarnos el máximo. Se cambiaría el superíndice $\min$ por $\max$.

Por último, hay que tener presente como estas restricciones son susceptibles de ser combinadas en proyectos reales.

II.d. Tipos de recursos

Como ya comentamos, las actividades requieren diversos recursos para su ejecución. La buena planificación de estos recursos es algo esencial en la programación del proyecto, puesto que no prever la necesidad de ellos en cada hito de proyecto podría suponer la paralización de éste, o bien, considerar una sobreestimación de éstos, traería gastos innecesarios (por ejemplo de stock, en el caso de recursos materiales) que serían atribuidos a nuestro proyecto a todos los niveles. Por tanto, dentro de la programación se deberá decidir: qué tipos de recursos se van a usar en ese proyecto, en qué cantidad va a ser usado cada uno, y saber la estimación de consumo a lo largo de toda la vida del proyecto.

En la bibliografía encontramos tres categorías principales de recursos:

- **Recursos renovables:** están disponibles en su totalidad en cada periodo base, es decir, la cantidad de ese recurso se renueva cada periodo, y por tanto, sólo está restringido el uso de recurso en cada instante.

  El conjunto de recursos renovables se denota por R^{ρ}. Algún ejemplo claro serían los trabajadores, las herramientas, las máquinas… (por ejemplo, si consideraras como periodo en la programación un día, en principio, son recursos que estarán constantes y disponibles diariamente).

  Para saber el número concreto de unidades disponibles del recurso k en cada periodo t usaremos la notación a_{kt}^{ρ}.

El algoritmo propuesto en el artículo que analizamos en el presente proyecto, considera como hipótesis inicial que todos los recursos son de este tipo, lo que facilita en parte la programación, ya que es la situación más favorable en cuanto a disponibilidad de recursos.

- **Recursos no renovables:** tienen una limitación de consumo para el total del proyecto. Un ejemplo evidente es el caso del dinero: el proyecto tendrá una estimación de presupuesto admisible, y que en el mejor de los casos, no se debería sobrepasar.
  El conjunto de recursos no renovables se denota por R^{ν}, y la disponibilidad asociada del recurso k sería a_k^{ν}.

- **Recursos doblemente restringidos:** supone la combinación de los dos recursos anteriores, donde además de la restricción por periodo, también la tiene para el proyecto completo.

Existen otras clasificaciones menos representativas. Por ejemplo, también podemos encontrar recursos parcialmente renovables (o no renovables), menos restrictivos donde la disponibilidad del recurso está definido para un conjunto de periodos.

II.e. Estimación de la duración de actividades

Para un buen análisis y la obtención de conclusiones útiles en cualquier estudio, es imprescindible confiar en la fiabilidad de los datos que se utilizan, puesto que de otra forma, no tiene ningún sentido engañarse conscientemente y dedicar un tiempo en vano. Esta idea, en relación a la programación de proyectos, puede verse presente en muchos casos. Uno de ellos, y bastante evidente, sería en la estimación de la duración de las actividades que componen el proyecto. Por tanto, una estimación realista de estos tiempos de ejecución puede ser la clave para que una programación sea correcta o no.

Es importante tener en cuenta que estas estimaciones dependen del criterio de una persona, por ello, habrá que intentar buscar personas cualificadas, con experiencia y que puedan valorar honestamente qué tiempo se puede atribuir a esa actividad. Retomando el ejemplo de pintar una habitación, nadie mejor que el propio pintor para decirnos cuánto puede tardarse en colocar plásticos para proteger el suelo, dar una primera mano de pintura, dejar secar… etc. Por otra parte, buscar a una persona experta en la materia, lo que en principio hemos descrito como una ventaja, puede convertirse en un arma de doble filo, ya que puede perder objetividad. Por ello, sabiendo la responsabilidad de esta decisión, es habitual que haya supervisiones y correcciones en este aspecto, ya que condiciona toda la programación. También es cierto, que en proyectos de temáticas concretas, la experiencia pasada es la mejor ley a seguir, y supone nuestra mejor guía.

Otro problema, el cual tenemos siempre presente, es el riesgo intrínseco a un proyecto. En el momento en el que algo debe ser planeado antes de ser ejecutado, entra en juego la incertidumbre. Así, la decisión termina siendo una mezcla de razonamiento y, en el caso de duda, algo cuyo error no condicionara de forma embarazosa nuestro proyecto.

Habitualmente, la sobreestimación de los tiempos se ve ampliada conforme más niveles de control están involucrados en la estimación, es decir, si varias personas están presentes dentro de una actividad, tenderán a añadir cada uno "colchones" de tiempo por seguridad que se verán acumulados.

En relación a la obtención de los tiempos encontramos:

- **Duraciones deterministas:** se asume que la duración de la actividad se ha estimado con certeza. Dentro de esta opción podríamos valorar que sólo se puede realizar de un modo la actividad (tomaríamos el valor medio de su duración, pensando en una ejecución estándar, con los recursos necesarios, sin considerar posibles situaciones excepcionales) o bien, que hay posibles escenarios que condicionarían la duración (se tendrían en cuenta combinaciones de duraciones según la asignación de recursos; un pintor hace un trabajo en 10 horas, mientras que cuatro podrían tardar la cuarta parte en realizar la misma actividad).

- **Duraciones valorando la incertidumbre:** en este caso, se propone una opción estocástica que tenga en cuenta la probabilidad en las duraciones, de tal forma que no sean igualmente consideradas dos actividades que durando ambas 5 días, varíen entre 4 o 6, o entre 2 y 12 días. Una de las propuestas para el modelo PERT fue la consideración de tiempos optimistas, probables y pesimistas, y mediante una distribución beta, se calcula la media y varianza de los eventos de la red.
  Debido a la exclusividad de algunos proyectos, no se puede recurrir a historiales sobre duración de actividades, y como resultado no se pueden obtener distribuciones. En ese caso se trataría con imprecisión, más que con incertidumbre. Hay una teoría desarrollada, en lo que sería la traducción "números confusos", en la que se consigue trabajar con esta información más imprecisa.

II.f. Variedad de funciones objetivo

Si nos planteamos desde un primer momento cuál es el objetivo final de la programación de proyectos, diríamos que es la obtención de una programación factible que establezca los comienzos y finales de todas las actividades dentro de la planificación. La solución, además de cumplir

restricciones de precedencias y recursos, deberá conseguir, en la medida de la posible, optimizar la función objetivo que se haya fijado (que considere el project manager en cuestión).

Podemos encontrar multitud de objetivos en la programación de proyectos. Algunos de los más habituales y por otra parte, más lógicos, dentro de la ejecución de una cartera, son:

- **En relación al tiempo:** es muy habitual buscar una programación que minimice la duración total del proyecto. Algunas de las ventajas son evitar el riesgo de superar una fecha límite o tener, en definitiva, más tiempo para acometer futuros proyectos.

- **... a los recursos:** en este caso, consideraríamos que la mejor solución sería aquella que trajera un menor consumo de recursos, bien globales (para toda la cartera) o locales. En múltiples ocasiones puede tratarse de uno de los criterios más lógicos, puesto que los recursos empleados suponen una de las conexiones más directas con los costes totales.

- **... a la financiación:** si buena parte del proyecto va a estar financiado externamente, puede que nos interese conseguir que la programación quede planificada de tal forma que la financiación sea lo más económica posible. La planificación puede estar más orientada a que en determinados plazos de tiempo se hayan realizado hitos concretos del proyecto, de tal forma que podamos recibir concesiones, entradas de flujo anticipadas….

- **... a la calidad:** puede tratarse de una función objetivo más ambigua, puesto que el concepto de calidad está acompañado de cierta subjetividad. La idea es que, de alguna forma, se pudiera justificar que al programar un proyecto de una forma u otra, la calidad del resultado final fuera mejor.

Como cabe pensar, el mundo real y los proyectos reales son mucho más complejos, lo que hace que habitualmente no hablemos de una única función objetivo, si no de varias, lo que nos hace pensar en *objetivos múltiples*. De esta forma, algunos autores atribuyen un peso ponderado a varios objetivos, de tal forma que podemos incluir varios y dándoles la importancia relativa que el manager considere.

III. CLASIFICACIÓN DE LOS PROBLEMAS DE PROGRAMACIÓN

Como ya hemos anunciado, el importante desarrollo y estudio del Project Scheduling en los últimos años, ha traído consigo una gran variante de problemas que emulan diferentes carteras de proyectos, donde las principales variables son el tipo de restricciones de recursos, el total de actividades o la función objetivo en cuestión.

En este apartado, vamos a plantear cuáles son las clasificaciones más habituales que encontramos en la bibliografía (problemas mono-proyecto / multi-proyecto), las características de cada uno y algunos de los métodos de resolución más implementados.

III.a. Problema de programación mono-proyecto (RCPSP)

Como su propio nombre indica, son problemas que involucran a un único proyecto, por ello es analizado en la literatura con anterioridad al resto. Reiteramos como, la programación de proyectos surge de la necesidad de optimizar y gestionar el uso de recursos, tiempo, , y es lógico, por otra parte, que una nueva tendencia comience desde las propuestas más simples.

Las siglas responden a la traducción inglesa Resource-Constrained Project Scheduling Problem, ya que habitualmente son una serie de actividades limitadas únicamente por el consumo de recursos, con las reglas de precedencia y el tiempo de finalización definidos.

Su supuesta sencillez, no implica que su resolución también lo sea, de hecho, de por sí, se trata de un problema NP-hard (concepto definido en el capítulo anterior), lo que supone el uso de heurísticas y algoritmos variados. Los más representativos son los que planteamos a continuación:

Heurísticas basadas en asignación de prioridades

Su uso es muy habitual, de hecho, el artículo que suscita el presente proyecto, se basa en uno de ellos. El concepto básico en todos los casos es priorizar las actividades en función de algún valor o cálculo previo. La decisión de qué valor será el que ordene las actividades, forma parte del trabajo previo del project manager. Como veremos más adelante, una de las mejoras incorporadas al algoritmo original del artículo, modifica dicho criterio de prioridad.

Lo que en principio puede considerarse una buena virtud (el hecho de que un ente externo tome las riendas del funcionamiento del algoritmo), puede llegar a ser un arma de doble filo, ya que esta decisión condiciona a todos los niveles la calidad de la solución. Por ello, en ocasiones es preferible

tener criterios variables o que se modifiquen tras algunas iteraciones y así poder escoger la mejor opción.

Los métodos más conocidos dentro de este grupo son:

- **Método serie:** el utilizado en el artículo. Es el más sencillo a nivel teórico, simplemente consideramos un criterio (actividades con menos consumo, un ratio entre consumo y tiempo de duración, media entre sus fechas mínimas y máximas de ejecución según las reglas de precedencia…) y las ordenamos de mayor a menor prioridad según éste. De esta forma, iremos incluyendo las actividades una a una mientras vaya siendo posible (que no superemos el límite de recursos en cada slot, o que sobrepasemos el límite de tiempo).
  Conforme van programándose se van constituyendo dos grupos: las secuenciadas y las disponibles (aquellas que ya podrían ser incluidas porque sus precedentes lo están, pero que aún no lo hicieron).

- **Método paralelo:** el método serie y paralelo fueron propuesto en ambos casos en *Kolisch et al. , 2006* y son métodos ampliamente utilizados y con buenos resultados. En este caso, con un funcionamiento bastante similar al anterior, el método paralelo valora cada actividad dentro del grupo de disponibles para cada slot y procede a colocarlas hasta que ya no quede ninguna, o no sean programables por superar el límite de recursos. Por tanto, el grupo de disponibles se recalcula para cada slot y terminamos haciendo un barrido completo al makespan del proyecto.

- **Método backward-forward:** uno de los primeros métodos, propuesto por *Li y Willis, 1999.* A diferencia de los anteriores, en este caso no generamos soluciones, si no que las consigue mejorar, por lo que es ampliamente utilizado como complemento de alguno de los métodos anteriores. La principal ventaja es que consigue reducir considerablemente los tiempos de ejecución.
  Se basa en el concepto de las holguras (el tiempo que se puede retrasar la programación de una actividad sin que por ello retrase a ninguna otra. En el caso de que una actividad tenga holgura 0 significaría que necesariamente tiene que comenzar en ese slot, no tiene otra opción). En primer lugar, partiríamos de una solución factible del problema. Una vez obtenida, haríamos una lista con las fechas en las que concluyen todas las actividades.
  En este instante comenzarían las dos operaciones que dan nombre al método: en la etapa *backward*, tomamos la actividad última y la retrasamos todo lo que su holgura permita, es decir, en este primer caso, hasta el valor máximo del horizonte temporal. Esto modificará las holguras del resto de actividades (aumentándolas). Así, una a una, desde la actividad que termina más tarde a la que termina más pronto, vamos haciendo esta operación. De alguna forma, conseguimos trasladar la programación hacia la derecha, retrasarla lo máximo posible.

En la etapa *forward*, se realiza un proceso similar, pero a la inversa. Tomando las actividades en el orden de menor a mayor fecha de inicio, y una a una, se intenta adelantar su inicio todo lo posible.

Visualmente, podemos pensar como el método backward-forward, lo que hace es "agitar" la solución factible que le proporcionamos para que quede realmente la mejor programación posible para esa propuesta.

Por último, presentamos algunas de las propuestas como criterios de prioridad encontradas en la bibliografía. Como vemos, son muy variadas y de primeras, no se puede saber cuál es la mejor opción, dependerá de cada cartera en cuestión:

Criterio	Autores	Variable
Número de act. sucesoras	*Álvarez-Valdés y Tamarit*	$\|S_j\|$
Fecha máxima de inicio	*Álvarez-Valdés y Tamarit*	$LFT_j - d_j$
Fecha más tardía de finalización	*Davis y Patterson*	LFT_j
Holgura mínima	*Davis y Patterson*	$LFT_j - EFT_j$

S_j: *conjunto de sucesoras*

LFT_j: (*Last Finish Time*): *última fecha posible de finalización de una actividad*

d_j: *duración de la actividad*

EFT_j: (*Early Finish Time*): *fecha más temprana de finalización de una actividad*

Como vemos, en la mayoría de los casos, las propuestas van en torno a las fechas posibles de programación de cada actividad o las duraciones. Ya anticipamos como el criterio de prioridad usado en la publicación tiene en cuenta el inicio que a la actividad le interesaría tener más la mitad de su duración. La idea es similar al concepto de centro de gravedad de la actividad.

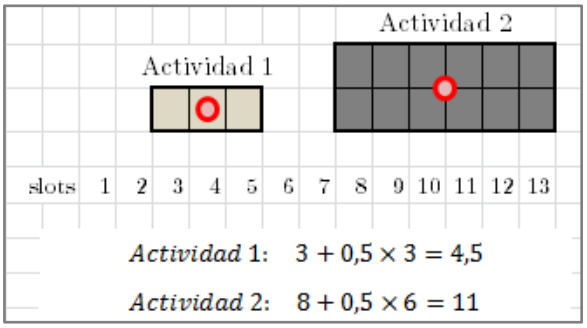


Figura. Ejemplo de cálculo del criterio de prioridad según la opción del artículo

Algoritmos de branch & bound

La traducción literal en español sería *Algoritmos de poda y corte*, y realmente, da una buena impresión del concepto que hay detrás de ellos. Son una alternativa muy importante en el caso de que tratemos con proyectos complejos donde la obtención de la solución exacta no es abordable computacionalmente, o bien, que por las características del proyecto, no nos interese saber exactamente cuál es la solución, si no que una aproximación sea suficiente, ahorrando esfuerzo computacional y, al fin y al cabo, tiempo. Es habitual en materias generales de optimización (no sólo en Project Scheduling) el uso de estos algoritmos, donde asemejamos el proceso a un árbol compuesto de múltiples ramas.

La dinámica parte del conjunto global de soluciones, de tal forma que se comienza a ramificar en subconjuntos independientes hasta que ya, gracias a algún criterio, sepamos que nunca se va a encontrar una solución mejor posible por ese camino. Algunos de los criterios de ramificación son:

- **Deph-first:** tras una ramificación, se crean uno o varios nodos que pertenecen al mismo nivel (es decir, todos ellos tienen el mismo nodo origen). Este proceso toma uno de esos nodos, y lo explota hasta que ya no hay más opciones de ramificar. Después, coge otro nodo de ese mismo nivel y repite la misma operación, así con todos ellos. Hace un barrido desde el nodo hasta completar todo su camino en el grafo.

- **Best-first:** en este caso, se va haciendo la llamada explotación por niveles. Se hace primero la ramificación de todos los nodos nivel 1, después de todos los nivel 2, que son sus "hijos", etc. Se trata de una mejor alternativa porque consigue filtrar desde un principio más soluciones (que con el deph-first sí que podrían considerarse candidatas). Como inconveniente, por otra parte, es el mayor peso en la computación, puesto que se están manteniendo durante más tiempo un mayor número de nodos activos.

Algoritmos genéticos

Se trata de una metaheurística presentada por *Holland, 1995.* Como su nombre nos suscita, tiene cierta vinculación con la idea de la selección natural, evolución biológica de las especies…, de tal forma que, "las soluciones que superviven son las mejores". Por ello partimos de la generación de una población de posibles soluciones, la cuál puede ser aleatoria, o bien utilizando algún método de los previamente explicados.

La diferenciación en los algoritmos propuestos en la bibliografía viene principalmente de la forma en la que se crea esta primera población. A partir de ahí, el procedimiento es muy similar: tomamos dos de las soluciones y haciendo una combinación de ambas obtenemos una nueva solución

a la que mutaremos. Una vez tengamos un número considerable de soluciones, nos quedamos con las mejores, y de nuevo repetiremos el proceso de combinación y mutación.

Con esta dinámica vemos como, evidentemente, conforme repetimos iteraciones, vamos mejorando la "raza" de las soluciones hasta llegar a una buena aproximación de la solución óptima.

Un ejemplo de algoritmo genético son los llamados algoritmos meméticos (*Ángela H.L. Chen y Chiuh-Cheng Chyu, 2008*). La aportación o variante que proponen es que, antes de la combinación de dos soluciones, éstas pueden haberse modificado o haber evolucionado debido a otras razones más allá que sólo la herencia genética. Es decir, los descendientes heredan otros comportamientos o recuerdos de sus soluciones previas. Esto consigue una mejora en las soluciones obtenidas, puesto que considera más variables, lo que en muchas ocasiones termina siendo positivo.

III.b. Problema de programación multi-proyecto (RCMPSP)

Si bien entendíamos como lógico el comienzo de la investigación de proyectos (y de cualquier rama de la ciencia) por los casos más sencillos, en donde involucrábamos un único proyecto, sigue siendo lógico que la evolución en la rama sea ir aumentando la complicación de los problemas a resolver. De ahí, es entendible este nuevo grupo de problemas RCMPSP (Resource-Constrained Multi-Project Scheduling Problem).

En grandes empresas, y desde una visión más realista, es habitual que convivan varios proyectos en el tiempo, y que por ello, tengamos que optimizar no sólo uno, si no varios. Partiendo de esta misma base, existe una clara clasificación entre lo que se denominan problemas centralizados (CRCMPSP) y descentralizados (DCRCMPSP).

El matiz que atribuye el problema centralizado es considerar a todos los proyectos como un gran proyecto en el que hay que secuenciar un mayor número de actividades (las del conjunto de proyectos). Este punto de vista no deja de ser el más ligado al sentido común, o al menos lo que antes se plantearía una gran mayoría de personas, por ello fue de los primeros enfoques antes el problema multi-proyecto. Por otra parte, es cierto que esta simplificación tiene claras desventajas (el hecho de que los proyectos no tengan gran capacidad de decisión en el proceso, o que no se tenga en cuenta las aportaciones individuales de cada uno, que irremediablemente hace que unos proyectos sean más interesantes que otros).

En el caso del planteamiento descentralizado, nos tenemos que situar a solamente hace unos años (Confessore, 2006). Se incluyen por primera vez sistemas multiagente, de tal forma que la conveniencia de un proyecto para programarse o no, depende directamente de la solución global que se plantee. Ya sí que se considera a cada proyecto de forma indivual y el que la solución global sea óptima depende de la conveniencia de esa solución para cada proyecto. Para ello, en los problemas

descentralizados se incluyen objetivos locales a nivel proyecto, pero también objetivos globales a nivel cartera.

Problema centralizado multi-proyecto (CRCMPSP)

Como ya anticipábamos, en el caso centralizado, tratamos a todos los proyectos globalmente sin atender a sus preferencias individuales. Este tratamiento hace que las soluciones obtenidas sean interesantes como solución global, pero que quizás no satisfagan del todo intereses propios de cada proyecto (un ejemplo evidente sería el pensar que quizás un proyecto podría verse finalizado en su tiempo de camino crítico, y por programar o prevalecer actividades de otros, pueda concluir mucho más tarde).

Por todo lo que hemos comentado, es lógico pensar que los métodos propuestos para el caso mono-proyecto siguen siendo igual de válidos en esta propuesta, y así es. La mayoría de las heurísticas de asignación de prioridades, por ejemplo, tienen su equivalente en el problema multi-proyecto.

Las posibles complicaciones vendrían de este trato global a todos los proyectos, ya que podríamos encontrar que cada uno tuviera objetivos individuales que no se tienen en cuenta a priori. Esto hace que, en ocasiones, la solución que nos ofrezcan los métodos ajustados desde el estudio mono-proyecto, llegue a ser errónea o incluso inviable.

Por ello, el verdadero éxito del planteamiento como CRCMPSP vendrá cuando los objetivos locales de los proyectos no sean del todo importantes y no puedan llegar a condicionar la solución definitiva.

Algunas publicaciones que presentan lo que hemos comentado son las propuestas por *Liu y Zheng, 2008* o *Lova y Tormos, 2000*.

En el primer caso, hablamos de una simple modificación del método paralelo, en el que como lista de actividades, tenemos las actividades de todo el conjunto de proyectos. Aún con soluciones admisiblemente buenas, proponen la combinación con una heurística de refinamiento.

En el segundo caso, apuestan por una combinación de las heurísticas de asignación de prioridad en función de cuál sea el objetivo principal a optimizar. En la siguiente imagen, perteneciente a la publicación, vemos claramente su propuesta.

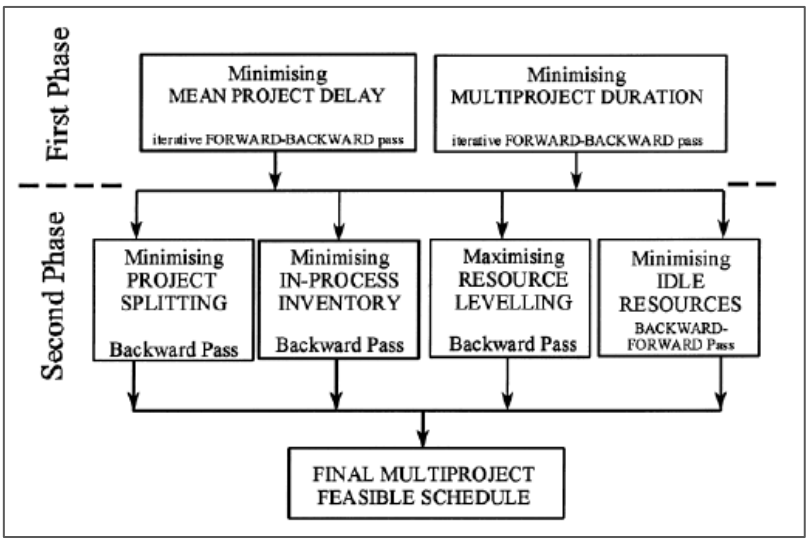


Figura. Modelo para CRCMPSP propuesto por *Lova y Tormos, 2000.*

Partiendo de una población de soluciones (a través del método serie o paralelo, por ejemplo), en una primera fase, aplicamos el método backward-forward sea cual sea nuestro objetivo final. Sin embargo, en una segunda fase, dependiendo qué objetivo nos interese más, aplicamos una heurística u otra, para finalmente obtener la programación factible multi-proyecto que buscábamos.

Como vemos, queda presente el hecho de que no favorecemos a ningún proyecto respecto a otro, ni tenemos en cuenta variables de ningún tipo sobre ellos (tratamiento centralizado). Quizás, este sesgo lo podríamos incluir al generar la población inicial de soluciones, donde ya la regla de prioridad que condiciona la programación favorezca a unos proyectos respecto a otros.

Como ya comentamos, la decisión de la regla de prioridad es un punto crítico en el éxito de la programación, por lo que en la bibliografía también encontramos publicaciones que estudian cuáles son las mejores reglas de prioridad en función del problema con el que estemos tratando.

Problema descentralizado multi-proyecto (DCRCMPSP)

La característica principal para el problema descentralizado es que, todos los proyectos compiten por la concesión de una cantidad de recursos que deben ser compartidos entre todos, de ahí que para cada instante tendremos una demanda acumulada de uno o varios recursos de los proyectos que coincidan simultáneamente en el tiempo.

Ya dijimos que en esta ocasión estaríamos en un sistema multi-agente, en el que cada proyecto tendrá un agente director que solicite los recursos oportunos a un agente coordinador (o ente superior) que intentará satisfacer todas las posibilidades mientras sea posible. Dentro de todo el

horizonte temporal, habrá instantes donde sea posible y en otros donde la capacidad será insuficiente. Uno de los grandes inconvenientes radica en maximizar el beneficio de los proyectos individualmente, además del propio a la organización.

Una de las propuestas con mejor acogida actualmente y de las más recientes para discriminar qué proyectos se ejecutarán (recibirán los recursos necesarios por parte del ente superior) y cuáles no, es la simulación de mercados. La idea es que, de alguna forma, los proyectos luchen por conseguir los recursos, persistiendo aquellos que tengan un mayor interés en programarse.

Por ejemplo, en el artículo de referencia para este proyecto, se plantea como alternativa para la simulación de mercados un método de subasta. De esta manera, los proyectos solicitan los recursos que necesitan en el tiempo según su programación óptima. El agente coordinador, al ver superada su capacidad respecto a dicha demanda global, modificará un hipotético precio de los recursos en cada slot (en donde haya mayor demanda, el precio subirá, mientras que donde sobren recursos disponibles, el precio bajará). Así, conforme se sucedan las iteraciones, comenzará a haber un desplazamiento de proyectos a lo largo del makespan y habremos conseguido la priorización que buscábamos: cuáles nos interesan acometer antes (con el objetivo de maximizar el valor total de la programación) y cuáles se pueden ver más relegados a lo largo del horizonte.

Cabe preguntarse dentro del método de subasta que hemos comentado, qué es lo que incorpora el concepto de subasta combinatoria presente en este proyecto fin de carrera. La explicación es sencilla. Cuando un proyecto necesita más de un tipo de recursos, las necesidades de sus actividades van a ser múltiples, una combinación de varios recursos. No tendría sentido que el ente superior concediera uno de los recursos y otros no (o todos y se puede programar la actividad, o ninguno), por ello, las pujas por parte del proyecto son combinadas, de ahí que la subasta se complemente con el adjetivo combinatoria.

Quién fue pionero en esta idea de introducir las subastas combinatorias fue *Confessore, 2006*. Además de retomado por múltiples autores, ha servido como referencia para otros modelos de selección y programación de proyectos.

III.c. Problemas de selección y programación de carteras de proyectos (PPSSP)

Se trata del último gran punto de trabajo en torno a la rama del Project Scheduling. Si la idea en el caso multi-proyecto programaba las actividades de todos los proyectos (bien desde el punto centralizado o descentralizado), como bien indica el título del presente apartado, ahora incluímos el poder de selección.

La reflexión sería la siguiente: si tenemos un conjunto de proyectos que conforman una cartera, y queremos saber cuándo ejecutar sus actividades según el interés del valor global de la solución,

implícitamente ya estamos obteniendo qué proyectos son más interesantes de ejecutar, luego, ¿por qué no usar esta información para seleccionar qué proyectos acometer y cuáles son susceptibles de rechazar? Por ello, en los problemas PPSSP, marcando un horizonte temporal máximo, conseguimos esta selección: aquellos proyectos que hayan concluído hasta esa fecha serán interesantes para mi organización, mientras los que no se hayan programado o no consigan incluir todas sus actividades, serán descartados de nuestra subcartera final.

Pues bien, la propuesta en el artículo *"Combinatorial auction algorithm for project portfolio selection and scheduling to maximize the net present value"* (Shou & Huang, 2010) que suscita este PFC, se trata de un método de resolución para este tipo de problemas. Al ejecutar una cartera de proyectos y aplicar el algoritmo en cuestión, se va realizando, iteración tras iteración, una puja individual de los proyectos según comentamos anteriormente en el método de subasta.

Así, al poner un horizonte máximo temporal, el método serie va programando todas las actividades posibles hasta que se alcanza ese límite. En cada iteración tendremos, proyectos que han entrado en la subcartera solución, y proyectos que han sido descartados por el propio método. Esta solución tendrá asociado un valor global de la programación (los proyectos tienen un valor máximo si se realizan en su tiempo de camino crítico, o menor valor conforme se alejan de esta fecha), de tal forma, que nos quedaremos con aquella solución que aporte un mayor valor a la organización, habiendo respetado el máximo de recurso global en cada slot, respetando las reglas de precedencias y terminando antes de la mencionada fecha límite (es decir, una solución factible y de máximo valor: optimización de la cartera en cuestión).

Anteriormente, sí que hubo intentos de elegir subcarteras pero dejando fija la programación individual de cada proyecto, es decir, considerar a cada proyecto como cajas negras que sólo se van a programar así. La ineficiencia de las soluciones, hizo que las nuevas tendencias sí que pusieran en juego el poder intercalar las actividades de todos ellos, y abriendo la posibilidad de más variaciones en las soluciones, lo que consiguió un mayor éxito a todos los niveles.

Uno de los algoritmos con más relevancia fue propuesto por *Chen y Askin, 2009.* Como particularidad, el paso inicial es numerar a los proyectos según el criterio más interesante para el decisor. Para ello, usan las llamadas *Fathoming Rules* en las que se da prioridad a los proyectos que son más factibles de encajar en la cartera añadiendo valor. Una vez ordenados, se establecen todas las enumeraciones de proyectos posibles, y se queda con la de mayor beneficio aplicando un algoritmo del tipo branch & bound.

Como cada proyecto se va añadiendo a la programación que se lleve hasta el momento (encajando en el tiempo y en los recursos disponibles que queden en cada slot), el principal inconveniente que posee es que, al no considerar todos los proyectos a la vez, se pueden llegar a soluciones no del todo óptimas.

Este defecto no estará presente en el algoritmo de nuestro artículo, ya que resuelve el problema de PPSSP considerando en cada iteración una nueva reestructuración de la organización. Como ya hemos comentado anteriormente, ésta irá variando en virtud del cambio de precios de los recursos y el interés individual de los proyectos.

En el próximo capítulo, estudiaremos de forma pormenorizada la propuesta del artículo, entrando en todos los arduos detalles matemáticos que respaldan la resolución.

IV. ANÁLISIS DEL ARTÍCULO

Una vez expuesto en los apartados previos en qué situación se encuentra el estudio de la programación de carteras de proyectos, procedemos a analizar en concreto el modelo que el artículo que suscita este proyecto fin de carrera nos plantea.

El artículo titulado *"Combinatorial auction algorithm for project portfolio selection and scheduling to maximize the net present value"* (Shou & Huang, 2010) desarrolla un algoritmo para abordar los problemas de optimización de carteras del proyectos que venimos describiendo hasta el momento. Según la clasificación que acabamos de abordar en el anterior capítulo, el modelo que se presenta entraría dentro del tercer gran bloque (PPSPP).

Desde una mirada crítica, podremos entender cuáles son las debilidades que tiene el algoritmo, lo que conseguirá fundamentar las mejoras que se han añadido en la reprogramación que planteamos.

IV.a. Descripción del problema

Se tiene un total de N proyectos y un conjunto de K recursos limitados (además, sólo son considerados recursos renovables, es decir, van a poder ser remplazables tras su uso, y en cada slot del horizonte temporal los tendremos disponibles). Entre dichos proyectos no hay ninguna conexión, exceptuando el hecho de que deben compartir esos recursos en sus programaciones. Concretamente, la cartera que queda resuelta en la publicación utiliza un único recurso (R1) con restricción para la cartera global de 3 unidades.

De forma individual, cada proyecto i tiene n_i actividades asociadas con sus respectivas precedentes y sucesoras. Para cada uno, se debería resolver el problema RCPSP (Resource Constrained Project Scheduling Problem) asociado, siempre bajo la optimización de la función objetivo considerada, que como ya vimos, puede ser de naturaleza muy variada (tiempo mínimo, mínimo coste de la ejecución del conjunto de proyectos, mayor beneficio…). En su caso, la mejor solución será aquella que consigue un mayor valor de la cartera programando el mayor número posible de proyectos.

Recordamos que durante todo el proyecto (tanto en el individual *i,* como en el conjunto de proyectos que se determine realizar) se deberán respetar:

- Las restricciones de recursos que nos vienen impuestas.
- Las relaciones de precedencia que hay entre las actividades de cada proyecto.

Es habitual que, con intención de conseguir la cartera de proyectos con mayor valor, se termine escogiendo un subconjunto dentro del total de los N proyectos disponibles, para obtener la cartera óptima. Es el tipo de problema que se aborda en el artículo y es conocido como PPSSP (Project Portfolio Selection and Scheduling Problem): se selecciona en primer lugar una subcartera de proyectos dentro de todas las opciones, y posteriormente se programan las actividades bajo las restricciones que ya conocemos. El criterio para obtener esta subcartera es definido a través de lo que denominan *Utilidad* del proyecto, que posteriormente describiremos.

Las variables a tener en cuenta en la formulación del modelo aparecen en la tabla.

Otro dato a tener presente, es que se supone que los flujos de caja asociados a la progresiva consecución del proyecto, son atribuidos al final de éste, cuando ya se ha completado.

Evidentemente, esta suposición no es muy realista. Este planteamiento hace que los resultados que pueda aportar el modelo, pierdan relevancia, puesto que la elección de unos proyectos u otros está totalmente condicionada a la financiación que se tenga durante el horizonte temporal en cuestión.

Notación	**Significado**
i	Índice del proyecto (i=1, 2,..., N).
j	Índice de la actividad dentro de un proyecto (j=1, 2,..., n_i).
t	Índice temporal (t=1, 2,..., T), con T como límite superior de la programación.
k	Índice del recurso (k=1, 2,..., K).
d_{ij}	Duración de la actividad (i, j), o de la actividad j dentro del proyecto i.
P_{ij}	Conjunto de actividades predecesoras de la actividad (i, j).
ST_{ij}	Start: Tiempo de comienzo de la actividad (i, j).
CT_{ij}	Completion: Tiempo de finalización de la actividad (i, j). ($CT_{ij}= ST_{ij} + d_{ij}$)
I_t	Conjunto de actividades en ejecución durante el periodo t.
R_k	Capacidad del recurso renovable k.
r_{ijk}	Cantidad del recurso k necesaria para la actividad (i, j).
$NPV_i(t)$	Valor neto del proyecto i si finaliza en el periodo t.

Tabla. Notación para la formulación del problema (Shou & Huang, 2010)

En el caso de problemas PPSSP se incluyen dos variables booleanas, que nos indican si un proyecto pertenece o no a la cartera (y_i) y si una actividad de un proyecto ha concluido en el instante t (x_{ijt}). Ambas variables valen 1 para el caso afirmativo, y 0 en el caso contrario. Es decir:

$$y_i = \begin{cases} 1, & \text{si el proyecto } i \text{ es aceptado,} \\ 0, & \text{si el proyecto } i \text{ es rechazado.} \end{cases}$$

$$x_{ijt} = \begin{cases} 1, & \text{si la actividad } j \text{ del proyecto } i \text{ está finalizada en el periodo } t, \\ 0, & \text{si no lo está.} \end{cases}$$

Gracias a esta última variable definida, es muy sencillo presentar la función objetivo a optimizar. Se trata de maximizar la suma del valor actual neto (*NPV*) de todos los proyectos. Como el NPV_i depende del periodo t en el que finalice el proyecto, x_{ijt} es 1 sólo en el instante t en el que su última actividad n_i termina.

La formulación del modelo de programación entera (Chen & Askin, 2009) es la siguiente:

$$\max \sum_{i=1}^{N}\sum_{t=0}^{T} \mathrm{NPV}_i(t)\cdot x_{i,n_i,t}$$

s.a.

$$R_k - \sum_{i=1}^{N}\sum_{j=1}^{n_i} r_{ijk} \sum_{\tau=1}^{t+d_{ij}-1} x_{ij\tau} \geq 0, \quad \forall k,t, \tag{1}$$

$$\sum_{t=1}^{T}(t-d_{ij})\cdot x_{ijt} - \sum_{t=0}^{T} t\cdot x_{iht} \geq 0, \quad \forall i,j,\ \forall (i,h)\in P_{ij}, \tag{2}$$

$$\sum_{t=0}^{T} x_{ijt} = y_i, \quad \forall i,j, \tag{3}$$

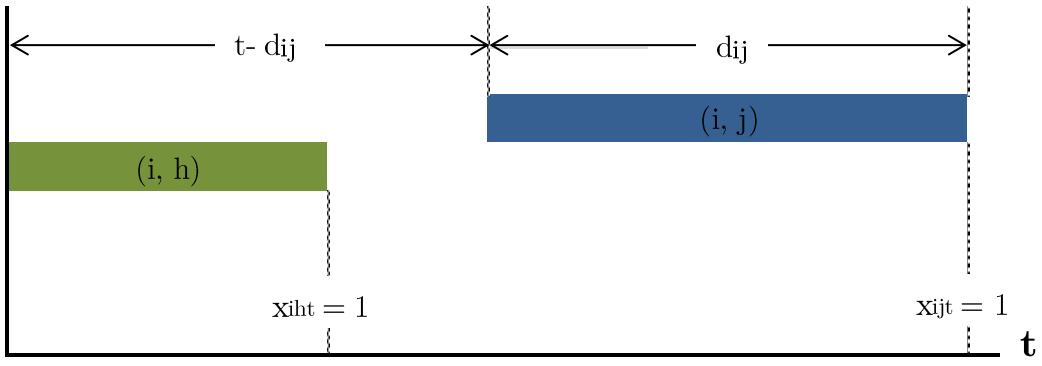
$$y_i \in \{0,1\}, \quad \forall i, \tag{4}$$

$$x_{ijt} \in \{0,1\}, \quad \forall i,j,t, \tag{5}$$

Si hacemos un análisis de lo que suponen esas cinco restricciones tendríamos:

(1) Restricción en recursos: no se pueden consumir en la ejecución de toda la cartera más cantidad de recurso renovable que la disponible. Es la única restricción que involucra a todos los proyectos con el sumatorio en *i*.

(2) Cumplimiento de las precedencias: según las reglas de precedencia de las actividades en cada proyecto, la programación será de una forma u otra, pero siempre respetándolas. Con este gráfico de ejemplo entre las actividades *(i, j)* e *(i, h)*, perteneciendo esta última a las precedentes de *(i, j)* (P_{ij}), queda más clara esta ecuación:



Como vemos, el instante $t-d_{ij}$ cuando termina (i,j) debe ser superior al instante en el que termina (i,h) para que realmente se estén respetando las precedencias.

(3) Asegura que ninguna actividad es preferente.

(4) La variable y_i es booleana, el proyecto es aceptado o no.

(5) La variable x_{ijt} es booleana, la actividad (i, j) ha terminado en el instante t o no.

IV.b. El concepto de subasta combinatoria en el modelo

El mecanismo de subasta es un método útil para dar precio a los recursos de los proyectos y poder seleccionar/programar carteras de proyectos, ya usado en muchas aplicaciones reales. En general, es un método intuitivo para resolver problemas clásicos de asignación de recursos.

Un procedimiento de este tipo es robusto con la presencia de dos niveles de control: un "coordinador superior" que es responsable de adjudicar los recursos a cada proyecto de la cartera (dependiendo de su puja en la subasta), y por otra parte, los project managers responsables de la programación y planificación de cada proyecto individual.

El hecho de que se hable de subasta "combinatoria" es debido a que posiblemente un proyecto no está interesado en un ítem en particular, si no en una combinación de varios de ellos, y por ello la puja vaya asociada a un pack de recursos (está interesados en todos ellos para poderse programar, o en ninguno y sale de la subasta). La aparición de este concepto surge en publicaciones en 2003 (Vries y Vohra) o en 2007 (Abrache et al.).

Es habitual que los valores de los ítems sean dependientes unos de otros, lo que hace que diferentes combinaciones tengan adjudicados diferentes valores. La ventaja de las subastas combinatorias es que consiguen soluciones (asignaciones de recursos a los pujadores) más eficientes en todos los aspectos.

Las aplicaciones de subastas combinatorias son muy diversas, no es algo propio del Project Management. Así, se han utilizado en distribuciones de tiempos en aeropuertos o en métodos de regulación del transporte. Eso sí, el concepto general es el mismo: un gestor superior oferta unos recursos y los postores proponen ofertas combinadas según su interés. La subasta se resuelve con la mayor eficiencia y el mayor beneficio para el coordinador.

Antes de abordar cómo aparecen las subastas en nuestro modelo, vamos a comentar brevemente qué otros tipos de subastas podemos encontrar en la literatura y sus principales diferencias.

Tipos de subastas tradicionales

Como ya sabemos, una subasta es un método de venta organizada basado en la competencia directa de los postores, es decir, quien más ofrezca por los artículos en venta, será el ganador. Si habitualmente estamos acostumbrados al método más clásico, en el que se conoce el precio superior

hasta el momento y se sigue superando la oferta hasta encontrar al pujador ganador, también podemos encontrar subastas en sobre cerrado, donde no hay oportunidad de mejora de la oferta en rondas sucesivas, y directamente se obtiene al adjudicatario.

Tradicionalmente encontramos, por tanto, subastas en sobre cerrado o subastas dinámicas (en versión inglesa, holandesa o americana, como desarrollaremos a continuación), aunque también mencionaremos otras nuevas opciones.

- **Subastas en sobre cerrado:** como su propio nombre indica, todos los postores entregan su oferta en una única ocasión de forma oculta, de tal forma que al desvelar las ofertas, gana quien ofreciera la mejor puja. Existe la versión de segundo precio, en la que el ganador, en vez de pagar lo que él había pujado, paga la segunda mejor apuesta de toda la subasta.

- **Subasta dinámica:** el postor ganador se obtiene tras rondas sucesivas en las que se va permitiendo el cambio de puja al conocer las ofertas de los otros pujadores.
  Dependiendo del sentido de la subasta diferenciamos entre tres tipos de subastas dinámicas: *subasta inglesa* (o ascendente, se va superando un precio de reserva con las pujas), *subasta holandesa* (o descendente, se parte de un precio determinado muy alto el cuál se va bajando, de tal forma que el ganador es el primero que acepte la puja en ese turno) y *subasta americana* (en la que todos los postores deben pagar la oferta que hicieron, pero el bien sólo lo consigue aquel que pujó con la cifra más alta).

- **Subasta Round Robin:** los postores emplazados son ordenados, de tal forma que el primero hace su puja. El segundo, al conocerla, puede superarla o bien salirse de la subasta. Se va realizando sucesivamente entre los pujadores hasta que sólo queda el postor ganador, aquel que ha superado todas las pujas del resto o bien, ha conseguido que ningún rival supere su oferta.

- **Subasta a la baja:** es habitual en subastas de obras o licitaciones en el servicio público, siendo ganador aquel que realice la obra por una menor cantidad.

Si volvemos a retomar el artículo, el funcionamiento en líneas generales es el de un subastador anunciando los precios de los objetos en subasta y los postores le proponen sus pujas combinadas en función de sus preferencias. Acorde a este concepto, aparecen nuevas variables en el modelo asociadas a la subasta interna de los recursos.

El recurso k en el periodo t es considerado como un solo artículo denominado g_{kt} (aunque es divisible y está formado por R_k unidades). Así, trataremos con la variable G como el conjunto de todos los g_{kt}:

$$G = \left\{ g_{kt} \,/\, 1 \le k \le K,\ 0 \le t \le T \right\}$$

Por otra parte, cada actividad requiere un subconjunto de recursos durante su ejecución. B_{ijk} es la demanda de recurso k de la actividad (i, j) durante su desarrollo, de modo que, para el artículo g_{kt}, el número de unidades requeridas por la puja B_{ijk} es exactamente r_{ijk}. De igual forma se puede definir B_{ij} como una puja combinada para el total de recursos usados por la actividad (i, j) y B_i como el conjunto de pujas combinadas para todo el proyecto i:

$$B_{ijk} = \left\{ g_{kt} \,/\ \mathrm{ST}_{ij} \le t \le \mathrm{CT}_{ij} \right\} \longrightarrow B_{ij} = \bigcup_{k=1}^{K} B_{ijk} \longrightarrow B_i = \bigcup_{j=1}^{n_i} B_{ij}$$

Pero esta última puja para el proyecto (B_i) puede ser que no sea factible. Recordamos que la programación de un proyecto era factible:

- **En cuanto a sus precedencias**, si se requería el comienzo de una actividad tras haberse realizado todas sus predecesoras. Si tenemos en cuenta que cada puja combinada para una actividad B_{ij} exigirá unos tiempos de comienzo de actividades concretos, la restricción de factibilidad de precedencias se puede escribir como:

$$\mathrm{ST}(B_{ij}) \ge \ \mathrm{ST}(B_{ih}) + d_{ih},\ \forall j,\ \forall (i,h) \in P_{ij}$$

  Es decir, el tiempo de comienzo de la actividad (i, j) según la puja B_{ij} debe ser posterior a la finalización de todas sus actividades predecesoras, que comienzan cuando su puja B_{ih} lo permite y tienen como duración d_{ih}.

- **En cuanto al uso de recursos**, si para todo instante de tiempo de la programación, no se sobrepasaba el límite de ningún recurso k. Se define la variable $D_{kt}(B_i)$ como el total de unidades del artículo g_{kt} requeridas en la puja del proyecto i, luego, la condición original la podemos transcribir como:

$$D_{kt}(B_i) \le R_k,\ \forall k, t,$$

  Es decir, no se pueden solicitar más unidades de recurso que las disponibles, R_k.

Cuando ya consideramos el conjunto de proyectos, hablamos de una cartera de proyectos, con su puja vinculada:

$$B = \bigcup_{i=1}^{N} B_i$$

donde de momento sólo requerimos que sea factible en cuanto a las precedencias. Es decir, esta puja de la cartera de proyectos (B) se considera factible si todos los proyectos que incluye son factibles según sus precedencias en las actividades. Por otra parte, se asume que ya entre proyectos no hay ninguna relación vinculante de precedencias, ninguno es necesario previamente para la ejecución de otro, algo que de nuevo, puede ser no muy fiel a la realidad en la programación de proyectos.

Aunque aún no debe ser factible en cuanto a recursos, la restricción de forma análoga es:

$$D_{kt}(B) \leq R_k, \ \forall k, t,$$

La demanda de recurso k en el instante t según la puja B, no puede ser superior a las unidades totales de recurso que disponemos.

IV.c. La función de utilidad de los proyectos

Definimos λ_{kt} como el precio del artículo g_{kt} en una ronda de la subasta iterativa. Luego, considerando el vector de precios λ en esa ronda, lo que se pagaría por la puja de un único proyecto sería:

$$P_i(B_i, \lambda) = \sum_{k=1}^{K} \sum_{t=0}^{T} \lambda_{kt} D_{kt}(B_i)$$

Como de momento hablamos de un único proyecto, podemos reescribir el precio como:

$$P_i(B_i, \lambda) = \sum_{k=1}^{K} \sum_{t=0}^{T} \lambda_{kt} \sum_{j=1}^{n_j} r_{ijk} \sum_{\tau=t}^{t+d_{ij}-1} x_{ij\tau}$$

Asociado a cada puja del proyecto, tenemos el valor actual neto (NPV) de éste según la programación obtenida, luego una función de utilidad sería la diferencia entre el valor que nos reporta el proyecto según esa puja y el precio que pagamos por los recursos:

$$U_i(B_i, \lambda) = NPV_i(B_i) - P_i(B_i, \lambda) = NPV(CT_{in_i}) - \sum_{k=1}^{K} \sum_{t=0}^{T} \lambda_{kt} \sum_{j=1}^{n_j} r_{ijk} \sum_{\tau=t}^{t+d_{ij}-1} x_{ij\tau}$$

Como vemos, la expresión se podría redactar como la diferencia entre el NPV del proyecto i cuando ha concluido su última actividad n_i y el coste de todas las unidades de recurso que ha necesitado cada actividad (al multiplicarse por la variable $x_{ij\tau}$, el coste total del recurso se computa en el instante en que se concluye esa actividad, que es cuando toma el valor de la unidad).

Una puja B_i con utilidad negativa no es admisible y no es tenida en cuenta, por ello, con ayuda de la función escalón (*sig(x)=1 si x>0*), se transcribe la utilidad de tal forma que sólo pueda tener valor positivo o nulo:

$$U_i(B_i,\lambda) = sig\Big(NPV_i(B_i) - P_i(B_i,\lambda)\Big)\cdot\Big(NPV_i(B_i) - P_i(B_i,\lambda)\Big)$$

Como en cualquier subasta, el interés del que puja es maximizar su beneficio, en este caso su utilidad, por lo que el problema que resuelve cada proyecto i de forma individual sería:

$$\max U_i(B_i,\lambda) = sig\ \left(NPV_i(B_i) - P_i(B_i,\lambda)\right)\cdot\left[NPV(CT_{in_i}) - \sum_{k=1}^{K}\sum_{t=0}^{T}\lambda_{kt}\sum_{j=1}^{n_j} r_{ijk}\sum_{\tau=t}^{t+d_{ij}-1} x_{ij\tau}\right]$$

s.a.

$$\sum_{t=1}^{T}(t-d_{ij})\cdot x_{ijt} - \sum_{t=0}^{T} t\cdot x_{iht} \geq 0, \quad \forall j,\ \forall(i,h)\in P_{ij},$$

$$\sum_{t=0}^{T} x_{ijt} = y_i, \quad \forall j,$$

$$x_{ijt} \in \{0,1\}, \quad \forall j,t,$$

Ya anticipamos que a este subproblema del pujador no le exigimos aún que cumpla las restricciones debidas al límite de recursos, simplemente tiene como objetivo minimizar flujos de caja negativos, y maximizar los positivos a lo largo de la vida del proyecto (siempre, eso sí, respetando precedencias de actividades).

Es necesario incluir en el modelo matemático estos costes c_{ijt}, donde como flujos de caja positivos (inflows) en el proyecto tenemos el NPV al concluir el mismo, y como negativos (outflows), los gastos en recursos a lo largo de las actividades del proyecto. Así:

$$c_{ijt} = \begin{cases} -\mathrm{NPV}_i(t) + \sum_{k=1}^{K} r_{ijk}\sum_{\tau=t-d_{ij}}^{t-1}\lambda_{kt} & si\ j = n_i \\ \sum_{k=1}^{K} r_{ijk}\sum_{\tau=t-d_{ij}}^{t-1}\lambda_{kt} & si\ j < n_i \end{cases}$$

Como vemos en la función a tramos descrita, mientras que no acaba la última actividad n_i no llega el valor del proyecto y sólo se incurre en los costes.

Al incluir los flujos y precios de los recursos, es lógico entender que resolver el problema de maximizar la utilidad, es equivalente a resolver el problema de minimizar los costes, por lo que la función objetivo también se puede ver como:

$$\min \sum_{j=1}^{n_i} \sum_{t=0}^{T} c_{ijt} x_{ijt}$$

Pensando en lo que representa, supone sumar los costes de las actividades de $(i,\ 1)$ a $(i,\ n_i)$ en el instante en el que se ven finalizadas.

En concepto, se trata de un problema para minimizar costes atendiendo a unas relaciones de precedencia, típicamente resuelto en la bibliografía convirtiéndolo en un problema equivalente de "flujo máximo". Se encuentran múltiples metodologías para afrontar el problema de flujo máximo en lo que termina siendo un diagrama de nodos y arcos. En este caso, se recurre al método Push-Relabel (Cherkassky & Goldberg, 1997) que será desarrollado con detalle a continuación y que presenta una gran eficacia computacional.

Método Push-Relabel para resolución del problema de flujo máximo

El cálculo del flujo máximo a través de un grafo esquemático compuesto por nodos y arcos, es un problema muy utilizado en múltiples estudios de investigación y campos muy variados, lo que ha hecho que numerosos algoritmos hayan sido desarrollados y que se haya dedicado mucha atención a optimizar y dar la mayor eficiencia computacional a este problema.

Una de esas opciones es el método Push Relabel (Cherkassky & Goldberg, 1997; Goldberg & Tarjan, 1988)*. Hasta que se presentó este método y el concepto del preflujo, algunos de los algoritmos más relevantes eran el de Ford y Fulkerson, el método de Dantzig, el algoritmo de Karzanov, el algoritmo de Tarjan o el del flujo blocante de Dinitz, este último superior en la práctica a todos los anteriormente mencionados. Pero es el método Push Relabel el que hasta el momento, es el más eficiente y rápido a la hora de afrontar este problema.*

En primer lugar, describiremos el método en cuestión con los conceptos que utiliza y la notación involucrada. Posteriormente veremos un ejemplo sencillo que demuestre de forma más intuitiva su funcionamiento.

Una red de flujo es un grafo directo $G = (V, E, s, t, u)$ *donde cada variable representa:*

V : conjunto de nodos. $n=|V|$
E : conjunto de arcos. $m=|E|$
s / t : fuente y sumidero de la red, respectivamente
u : capacidad admisible por el arco. Valor siempre positivo.

El arco que va del nodo v a w, se representa como (v, w) y además, se asume que existe el arco en sentido opuesto (w, v).

Un flujo es una función en los arcos que satisface las restricciones de capacidad en todos los arcos y las restricciones de conservación en todos los nodos, excepto en la fuente y el sumidero. Es decir, por un arco no puede pasar más flujo del admisible en su capacidad $u(v,w)$ y por otra parte, todo lo que entra en un nodo debe salir de él (excepto en s y t). De esta última restricción se define el exceso de un nodo v ($e_f(v)$) como la diferencia entre lo que entra y lo que sale de un nodo, por lo que la definición de flujo exige que el exceso sea nulo.

Un preflujo, concepto introducido por Karzanov ("Determining the Maximal Flow in a Network by the Method of Preflows", 1974), si bien también respeta las restricciones de capacidad en los arcos, admite que $e_f(v)$ pueda ser además de nulo, positivo. Es decir, la diferencia entre el flujo y el preflujo es que, en el último caso, se permite que en un nodo entre más de lo que sale de él.

En relación a los arcos, hablaremos de residuales o saturados. Un arco residual es aquel en el que su máximo flujo admisible aún no se ha alcanzado, mientras que en un arco saturado, como el propio adjetivo indica, no puede circular más flujo. En el caso de un arco residual, se define la capacidad residual $u_f(v,w)$ como esa cantidad en la que aún podemos incrementar el flujo que circula por el arco. Los arcos residuales constituyen el grafo residual G_f .

La distancia es una función $d: V \to \mathbb{N}$ que cumple que para el nodo final de la red $d(t)=0$ y para todo arco residual (v,w), $d(v) \le d(w)+1$. Se dice que el arco residual es admisible si cumple exactamente la igualdad de la última inecuación. De forma intuitiva, la función distancia sería equivalente a niveles de altura respecto a una referencia, que sería "el suelo" de la red, el nodo t. Por tanto, podríamos entender un grafo como una fuente (s) de la que cae agua por tuberías (arcos), con sus limitaciones, hasta llegar a su destino final en la base del grafo (t).

Como última definición básica, diremos que un nodo está activo si:

$$v \notin \{s,t\}, \quad d(v) < n \quad y \quad e_f(v) > 0$$

Luego, simultáneamente debe cumplir, que su distancia sea menor al número total de nodos de la red y que el exceso del nodo sea positivo (debe estar entrando más flujo del que sale de él).

El método Push-Relabel mantiene una distancia para cada nodo y un preflujo en la red. La inicialización del algoritmo tendría:

- *El preflujo nulo en todos los arcos.*
- *El exceso nulo en todos los nodos, excepto en la fuente, donde $e_f(s)$ representaría el flujo potencial de la red, es decir, la suma de las capacidades de todos los arcos.*
- *La distancia $d(v)$ como el mínimo entre el número total de nodos n y todas las distancias del nodo v al nodo final t a través del grafo residual G_f.*

Una vez definido el estado inicial de las variables, el algoritmo realiza las operaciones de actualización (Push y Relabel) según el siguiente pseudocódigo:

push(v, w).	
Applicability:	v is active **and** (v, w) is admissible.
Action:	send $\delta = (0, \min(e_f(v), u_f(v, w))]$ units of flow from v to w.
relabel(v).	
Applicability:	v is active **and**
	push(v, w) does not apply for any v.
Action:	replace $d(v)$ by $\min_{(v,w)\in E_f}\{d(w)\} + 1$.

Figura. Operaciones de actualización del método Push-Relabel (Cherkassky & Goldberg, 1997)

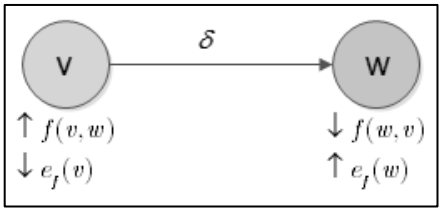


Figura. *Elaboración propia*

Para poder realizar la operación Push sobre un arco, el nodo v debe estar activo (distancia menor que el número total de nodos n y tener exceso de flujo) y además debe ser admisible ($d(v) = d(w) + 1$). En el caso de que se cumplan las dos condiciones, pasará una cantidad de flujo δ que, como es lógico, será el mínimo entre dos cantidades: el exceso del nodo (flujo que al nodo "le sobra") y la cantidad residual del arco (flujo que aún ese arco permitiría recibir antes de saturarse). En la imagen vemos las implicaciones que tiene en el preflujo y en el exceso el mover la cantidad δ de v a w. Diremos que la operación Push ha saturado el arco cuando lo deje con capacidad residual nula.

Para poder realizar la operación Relabel sobre un nodo v, debe estar activo y no debe haber posibilidad de realizar Push para ningún arco (v, w). Si es así, se modifica el valor de la

distancia de v por la distancia mínima de todos los arcos residuales que salen de v más la unidad. De esta forma estamos consiguiendo que el nodo se haga admisible para la siguiente iteración. Cuando un nodo ha recibido la operación de Relabel, inmediatamente se convierte en el primer nodo en el que hacer Push, ya que aseguramos que ya posee algún arco admisible en el que poder liberar el flujo sobrante.

Está comprobado como la eficiencia del método depende en gran parte del orden en el que se realicen estas dos operaciones. Para ello, entra en juego en el algoritmo otra función conocida como descarga del nodo.

Básicamente, el proceso de descarga insiste en hacer las funciones de Push y Relabel sobre un mismo nodo mientras este siga teniendo exceso positivo. De esta forma, de cada nodo se tiene un listado de sus arcos admisibles en cada instante y se repiten las dos operaciones mientras el nodo esté activo.

Pese a la eficacia del método Push-Relabel, hay que reconocer que el tiempo de ejecución dependiendo de las dimensiones de la red puede llegar a ser su mayor limitación, algo que ha quedado evidenciado al desarrollar este proyecto fin de carrera.

Para una mayor profundidad en la descripción del método remitimos a las publicaciones "A new approach to the maximum-flow problem" (Goldberg & Tarjan, 1988) *y "On implementing the Push-Relabel method for the maximum-flow problema"* (Cherkassky & Goldberg, 1997).

IV.d. El beneficio del subastador. Actualización de precios

Lo lógico es plantearnos que el subastador siempre va a pretender ganar lo máximo por los ítems vendidos, pero si consideramos que lo que recibe son los costes de los recursos, un aumento indiscriminado de los precios de los ítems, beneficiarían este objetivo. Por esta razón, en el caso de programación de carteras de proyectos, se persigue un concepto más vinculado a lo que sería el bienestar social: se busca la puja B que maximiza el NPV de todos los proyectos i que la conforman. Es decir:

$$\max_{B} \sum_{B_i \in B} \mathrm{NPV}_i(B_i)$$

Para encontrar esa puja que reporta el mayor valor a los proyectos de la cartera, se utiliza un mecanismo de precios. Se trata de una subasta combinatoria multi-unidad (MUCA), de gran

complejidad en su resolución, ya que cada proyecto no sólo depende de su utilidad, si no también de su estructura interna sometida al cumplimiento de factibilidad.

Para el caso de MUCAs es difícil alcanzar equilibrio entre demanda y oferta en una única ronda. El equilibrio se obtiene a través del llamado tanteo Walrasiano estándar, que en este caso funcionaría de la siguiente forma:

El subastador presenta los precios y acorde a ellos, los proyectos presentan sus pujas indicando qué cantidad de cada recurso demandarían. Para aquellos ítems donde hay exceso de oferta y el precio aún es positivo, bajarán los precios (para incentivar que en la siguiente iteración sean mejor considerados por los pujadores), mientras que para los que hay exceso de demanda, subirán, de tal forma que persistirán los proyectos más interesados en esos recursos. Hasta que no se llega al equilibrio tras sucesivas iteraciones basadas en este mecanismo, no se producen los intercambios.

Según esto, el precio λ_{kt} para una siguiente ronda, seguirá una expresión del tipo:

$$\lambda_{kt}^{b+1} = \max\left\{0,\ \lambda_{kt}^{b} + s\left(D_{kt}(B^b) - R_k\right)\right\}$$

de tal forma que si las unidades de k en el instante t que necesita toda la cartera ($D_{kt}(B^b)$) superan las disponibles (R_k, en el ejemplo resuelto de tres unidades) se ve incrementado el precio en ese valor por un paso fijo s.

Pero además de este método de actualización de precios en cada iteración, también se plantea otro más complejo, a través del tanteo Walrasiano adaptativo. Partiríamos de transcribir el modelo inicial de programación entera a una relajación Lagrangiana (LR).

Si recordamos, sólo la restricción asociada a los recursos (la primera de ellas), vinculaba a todos los proyectos de la cartera, mientras que el resto eran independientes a cada proyecto i. Así, con un multiplicador λ positivo, incluimos esa restricción en la función objetivo y el problema se transforma en:

$$\max\left[\sum_{k=1}^{K}\sum_{t=0}^{T}\lambda_{kt}R_k + \sum_{i=1}^{N}\left(\sum_{t=0}^{T}(\mathrm{NPV}_i(t)\cdot x_{i,n_i,t}) - \sum_{k=1}^{K}\sum_{t=0}^{T}\lambda_{kt}\sum_{j=1}^{n_i}r_{ijk}\sum_{\tau=1}^{t+d_{ij}-1}x_{ij\tau}\right)\right]$$

$$s.a.$$

$$\sum_{t=1}^{T}(t-d_{ij})\cdot x_{ijt} - \sum_{t=0}^{T}t\cdot x_{iht} \geq 0, \quad \forall i,j,\ \forall(i,h)\in P_{ij},$$

$$\sum_{t=0}^{T}x_{ijt} = y_i, \quad \forall i,j,$$

$$y_i \in \{0,1\}, \quad \forall i,$$

$$x_{ijt} \in \{0,1\}, \quad \forall i,j,t,$$

Si consideramos la función del modelo como una composición de dos subproblemas para los proyectos en solitario:

$$\upsilon(\mathrm{LR}_{\lambda}) = \sum_{i=1}^{N} \upsilon(\mathrm{LR}_{\lambda,i}) + \sum_{k=1}^{K}\sum_{t=0}^{T} \lambda_{kt} R_k$$

con $\upsilon(\mathrm{LR}_{\lambda})$ como el valor óptimo del problema para un λ dado,

$$\upsilon(\mathrm{LR}_{\lambda,i}) = \max\left(\sum_{t=0}^{T} (\mathrm{NPV}_i(t) \cdot x_{i,n_i,t}) - \sum_{k=1}^{K}\sum_{t=0}^{T} \lambda_{kt} \sum_{j=1}^{n_i} r_{ijk} \sum_{\tau=1}^{t+d_{ij}-1} x_{ij\tau} \right).$$

vemos como la expresión para $\upsilon(\mathrm{LR}_{\lambda,i})$ coincide con la función de utilidad que ya dijimos que cada proyecto debía maximizar.

$$\upsilon(\mathrm{LR}_{\lambda,i}) = \max U_i(B_i, \lambda)$$

Pues bien, la otra opción del paso variable en cada ronda b de la subasta combinatoria, viene propuesto bajo la expresión:

$$s^b = \alpha^b \frac{\upsilon(\mathrm{LR}_{\lambda^b}) - \mathrm{LB}}{\sum_k \sum_t \left(D_{kt}(B^b) - R_k \right)^2}$$

donde α_b es un escalar que varía su valor según el comportamiento de $\upsilon(\mathrm{LR}_{\lambda})$ en un número de rondas en la subasta, y LB es un límite inferior de la relajación Lagrangiana. Con esta elección de actualización, se consigue ajustar los precios sustancialmente en las primeras etapas del método.

IV.e. Cumplimiento de la restricción de recursos. Método en serie

Mientras no se llega al equilibrio del vector de precios y el método está iterando en su resolución, será habitual que la programación propuesta para la cartera no cumpla la factibilidad en cuanto al uso de recursos, siendo necesario una nueva heurística que consiga hacer la solución factible. Así, el método en serie (Kolisch, 1996) es una opción sencilla y eficaz para esta última transformación de la programación.

La base del método es que la prioridad de una actividad, depende del tiempo de comienzo de ésta en la puja de la cartera más una fracción en función de su duración total. Todas las actividades quedan programadas en serie, intentando siempre que las actividades empiecen lo antes posible sin violar el número total de recursos disponibles.

Evidentemente, tras aplicar el método en serie, es habitual que el makespan de la cartera se vea incrementado, incluso suponiendo que algunos de los proyectos involucrados vean reducido su $\mathrm{NPV_i}$

(recordamos que el valor del proyecto i dependía del instante en el que concluía en la programación). Por ello, es posible que finalmente haya proyectos que queden excluidos de la cartera tras este último ajuste.

En la siguiente figura podemos ver la demanda global de recursos sin ninguna restricción y la solución que propone el método serie en la ordenación de actividades según el criterio de prioridad de cada una. Las imágenes son obtenidas por la aplicación desarrollada en el presente PFC. Se trata de la primera iteración del algoritmo al resolver la cartera ejemplo que se propone en el artículo de Shou & Huang.

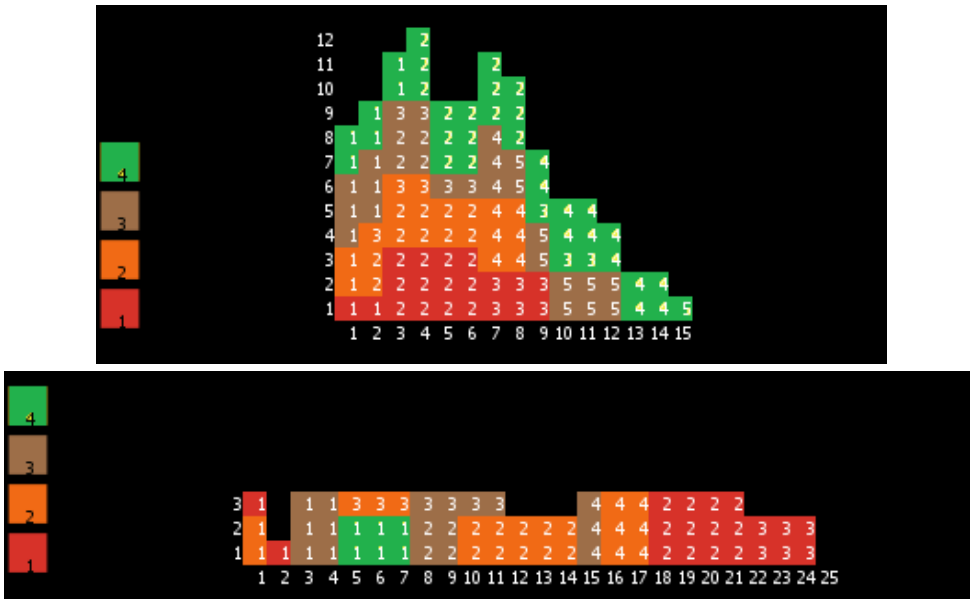


Figura. Salida gráfica de la programación en Netlogo desarrollada. Elaboración propia

Como vemos, no todas las actividades consiguen estar programadas, luego como advertimos, habrá proyectos que no finalizarán, de ahí que obtendremos una subcartera solución. La notación usada en la salida gráfica sería la siguiente: según el código de colores de los cuatro proyectos que aparecen a la izquierda, las actividades correspondientes a cada uno toman el mismo color. Por otra parte, el número que aparece dentro de cada rectángulo (que representa una unidad de recurso) se refiere al número de la actividad dentro de dicho proyecto.

Recalcamos como, mientras que sin restricción de recursos toda la cartera se programaría en un horizonte temporal de 15 slots, al restringir a tres unidades la disponibilidad, ni en un horizonte de 25 slots conseguimos programar los cuatro proyectos en cuestión. De ahí la importancia del límite de recurso global, por ello, una de las variables que modificaremos en nuestro análisis del algoritmo (capítulo VI), será este valor máximo admisible en cada slot.

El pseudocódigo que aparece en el artículo y que recoge la síntesis de todo el procedimiento del algoritmo sería:

Algorithm 1 Multi-unit combinatorial auction (MUCA) procedure
Input: d_{ij}, P_{ij}, r_{ijk}, R_k, $\mathrm{NPV}_i(t)$.
Output: y_i, x_{ijt}.

```
1   WHILE criteria NOT satisfied
2       FOR every single project i
3           SCHEDULE to maximize its utility;
4           COMPUTE its utility;
5           IF utility<0 THEN
6               SUBMIT no bid;
7           ELSE
8               SUBMIT a project bid;
9           ENDIF
10      ENDFOR
11      COMPUTE upper bound;
12      IF portfolio bid is NOT resource feasible THEN
13          CONVERT to a feasible portfolio bid;
14      ENDIF
15      COMPUTE lower bound;
16      UPDATE price vector;
17  ENDWHILE
18  OUTPUT the best portfolio (y, x).
```

Figura. Esquema básico del funcionamiento del algoritmo propuesto (Shou & Huang, 2010)

Como vemos, los datos de entrada (inputs) de los que parte la programación son la duración de las actividades (d_{ij}), información de las precedencias (P_{ij}), lo que consume cada actividad de cada recurso (r_{ijk}), el límite de unidades por cada recurso (R_k) y el valor del proyecto en función del tiempo en que concluya definitivamente ($NPV_i(t)$).

En el caso de salidas (outputs) sabemos si el proyecto ha sido incluido o no en la cartera (variable booleana y_i), y si es así, los tiempos en los que terminan sus actividades (x_{ijt}), que junto con las duraciones de éstas, nos permiten definir perfectamente la ejecución de la cartera en el horizonte temporal.

IV.f. Cálculo de límites superior e inferior (UB, LB)

Como acabamos de ver en el pseudocódigo del programa (líneas 4 y 15), en cada iteración se calcula un límite superior y otro inferior para la solución.

El límite superior (Upper bound, *UB*) se calcula tras obtener la subcartera de proyectos con utilidad positiva que optan a aparecer en la programación final. El valor del *UB* es la suma de los $NPV_i(t)$ de esos proyectos cuando finalizan, según sus intereses con respecto al precio de los

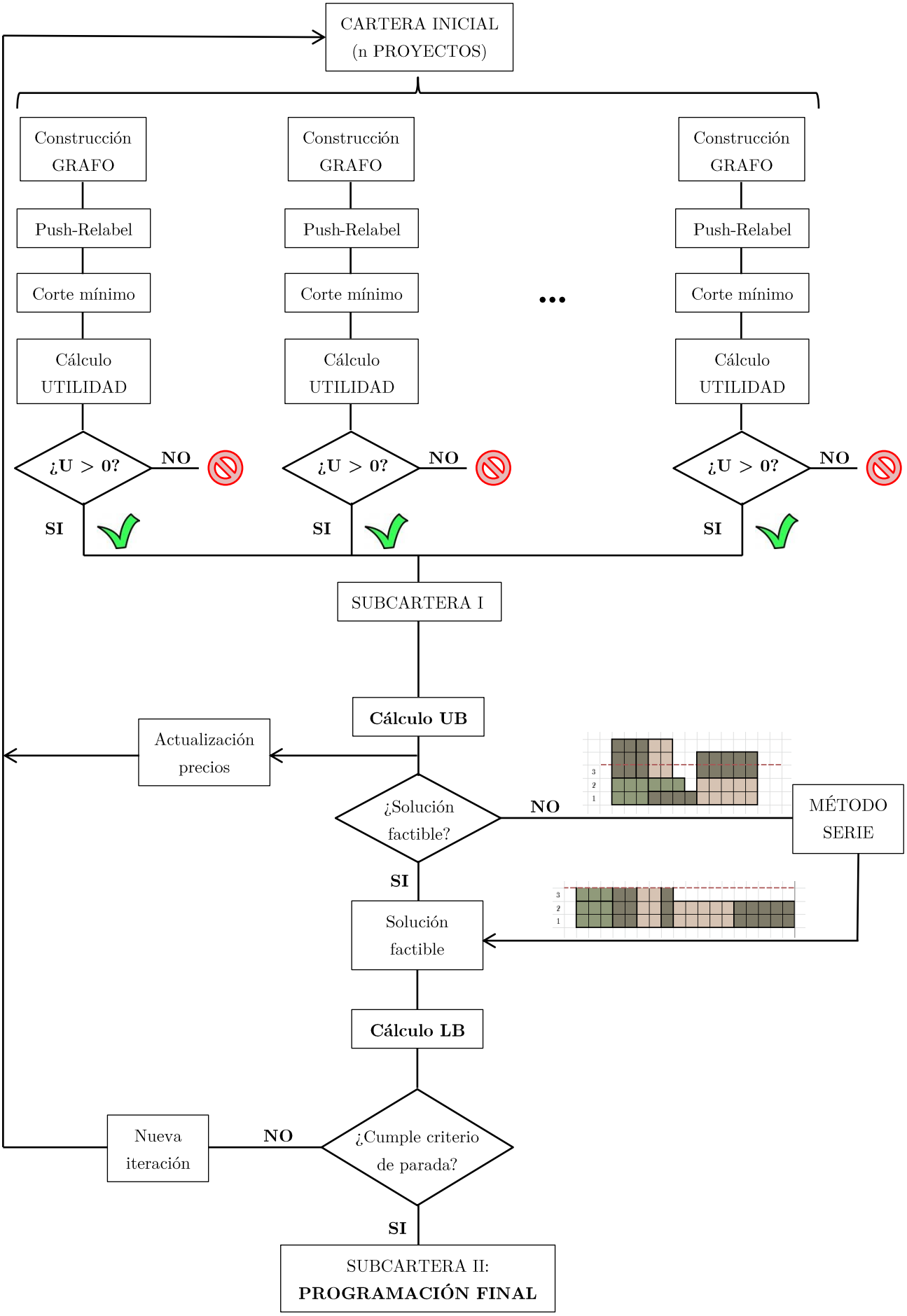
recursos en cada slot. Por ello, el valor máximo que puede alcanzar es la suma de los $NPV_{máx}$ de todos los proyectos, que ocurriría en la primera iteración (los precios comienzan inicializados a 0, y todos los proyectos deciden programarse según su camino crítico. Con utilidades positivas, todos quedan incluidos en la subcartera con su mayor valor posible).

Por otra parte, el límite inferior de la solución (Lower bound, LB) se calcula tras finalizar el método serie. Tomando sólo los proyectos que hayan conseguido programarse de forma completa, calculamos de nuevo su $NPV_i(t)$. Conforme más alejados del tiempo de camino crítico hayan terminado, mayor penalización tendrán sobre su $NPV_{máx}$.

En búsqueda de la solución óptima, límites superior e inferior deberían ir acercándose asintóticamente, de forma que la mejor solución, por definición, será aquella que posea el LB mayor, ya que habrá incluido más proyectos y más próximos a sus programaciones de camino crítico.

IV.g. Diagrama de bloques. Resumen de funcionamiento

A continuación, habiendo comentado el funcionamiento de cada una de las partes más significativas del algoritmo, pretendemos resumir en un diagrama de bloques cómo termina siendo el procedimiento en cada iteración. Presentamos en la página siguiente este esqueman general que clarifica al lector sobre las conexiones en la programación.


CARTERA INICIAL
(n PROYECTOS)
Construcción GRAFO
Push-Relabel
Corte mínimo
Cálculo UTILIDAD
¿U > 0?
NO
SI
Construcción GRAFO
Push-Relabel
Corte mínimo
Cálculo UTILIDAD
¿U > 0?
NO
SI
...
Construcción GRAFO
Push-Relabel
Corte mínimo
Cálculo UTILIDAD
¿U > 0?
NO
SI
SUBCARTERA I
Cálculo UB
Actualización precios
¿Solución factible?
NO
MÉTODO SERIE
SI
Solución factible
Cálculo LB
Nueva iteración
NO
¿Cumple criterio de parada?
SI
SUBCARTERA II:
PROGRAMACIÓN FINAL

Aprovechando el soporte gráfico, volvemos a resumir el funcionamiento general de la programación:

Partimos de una cartera inicial con n proyectos. Cada uno de ellos se resuelve de forma individual como comentamos: se realiza el grafo, se calcula el flujo máximo a través de la red (donde la capacidad de los arcos está vinculada a los precios de los recursos en esa iteración) por el método Push-Relabel, y por dualidad, ese flujo máximo nos indica el corte mínimo en la red, el cuál nos da la programación óptima para cada proyecto (sin tener en cuenta ninguna restricción aún).

De cada uno se calcula el valor de la utilidad (restamos al valor máximo del proyecto ($NPV_{máx}$) lo que cuesta esa programación según los precios en esa iteración, es decir, el flujo máximo que pasó por cada grafo). Si el valor de la utilidad es positivo, quedará incluído en la primera subcartera de proyectos (en el caso de que todos tengan U $>$ 0, la cartera íntegra podrá ser susceptible de completarse en la solución final). En ese momento se calcula el límite superior de la solución.

Una vez que traducimos las programaciones deseadas de cada uno en recursos solicitados, tenemos la demanda global de recursos. Con ella podemos actualizar los precios de los recursos en cada slot para la próxima iteración (los recursos en los slots más saturados aumentan sus precios, mientras que en los menos solicitados, disminuyen, para que así sean más atractivos para la próxima vez).

Por otra parte, es muy probable que esta programación conjunta no sea factible, es decir, que para algún slot se superen la cantidad límite de recurso global admitido. En ese instante es cuando entra en juego la heurística de asignación de prioridades, y el método serie va incluyendo las actividades según el criterio de prioridad y, ya sí, respetando el límite de recurso.

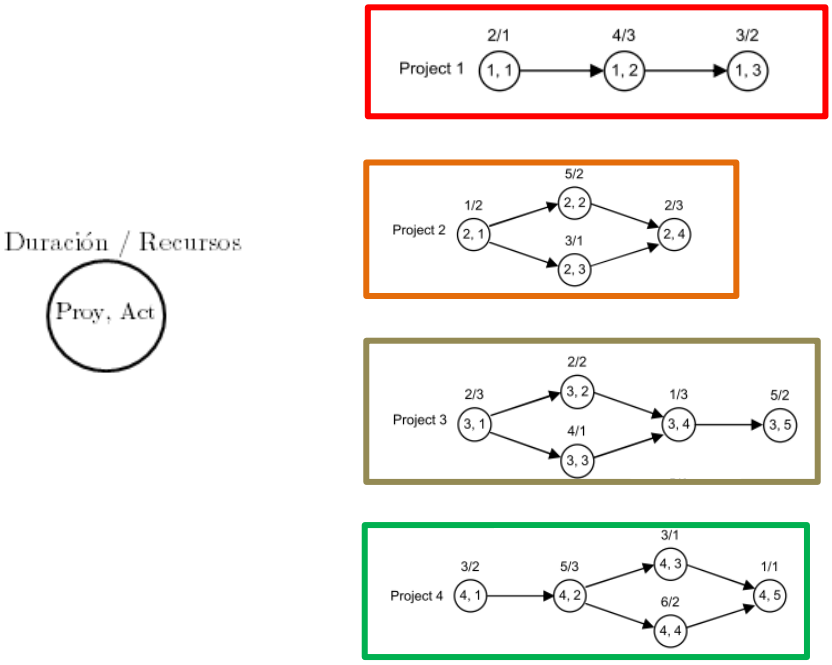
Como exite un límite de horizonte temporal, es posible que tras ejecutar el método serie, no todas las actividades de todos los proyectos de la subcartera terminen programadas, luego tendremos una subcartera II formada por los proyectos que sí que finalizan. De esta forma, conseguimos la que serie una solución factible para la programación. La búsqueda de la óptima supone la reiteración del método hasta que se cumpla el criterio de parada acordado.

IV.h. Descripción del problema resuelto en el artículo

Por último, como los proyectos que constituyen la cartera del problema que se resuelve en el artículo van a ser los usados en el análisis realizado en este PFC, vamos simplemente a recogerlos en este apartado, para además de tener los datos, poner de manifiesto cómo se trata de proyectos muy simples en cuanto a relaciones de precedencia y con muy pocas actividades. Esto demuestra que estamos abordando un modelo más teórico, en el que se pueda entender bien el funcionamiento interno del algoritmo, aunque esto nos aleje un poco de la verdadera realidad en la programación de

proyectos: un conjunto más numeroso y, en ocasiones, con cientos de actividades para cada uno de ellos.

La notación de colores respeta el tono que los agentes asociados a cada proyecto tienen en la aplicación en Netlogo. Los valores máximos que pueden reportar los proyectos si se programan según su camino crítico son 6, 8, 11 y 9 unidades respectivamente.



Por último, presentamos la solución que, al ejecutar esta cartera con un horizonte máximo de 25 unidades, se muestra en la publicación:

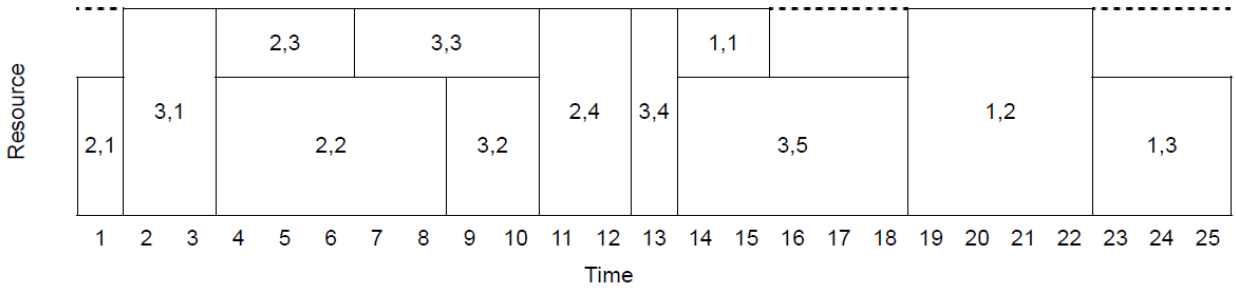


Figura. Solución del PPSSP propuesta en el artículo (Shou & Huang, 2010)

Como vemos, terminan finalizando el proyecto 2 en primer lugar (slot 12), el proyecto 3 (slot 18) y el proyecto 1, optimizando al máximo las 25 unidades de tiempo que comentábamos. Por tanto, la selección de proyectos rechaza al proyecto 4. La justificación anticipada que podemos dar, es que se trata de un proyecto más largo que los anteriores (si hacemos un cálculo sencillo en su grafo, vemos que su tiempo de camino crítico es de $3 + 5 + 6 + 1 = 15$, frente al 9, 8 o 12 de los

otros tres), y probablemente, el hecho de que su valor máximo no sea muy superior al resto, no le sitúa en una posición más favorable que contrarreste esa desventaja.

V. FUNCIONAMIENTO DEL PROGRAMA IMPLEMENTADO

Para la réplica del algoritmo en cuestión, se ha optado por el programa de simulación de agentes Netlogo. Si bien el punto fuerte y uso más habitual de este programa es la simulación de sistemas complejos que evolucionan con el tiempo, de los cuáles podemos extraer conclusiones relevantes sobre comportamientos en dichos sistemas que quizás estaban desconocidos hasta el momento, recurrimos a Netlogo por el gran poder visual que tiene. La interfaz que presenta permite visualizar fácilmente aquello que programamos. En nuestro caso, conseguiremos sacar por pantalla desde la resolución de los grafos de los proyectos con el algoritmo de Push-Relabel, hasta las demandas de recursos y programaciones finales en cada iteración.

Otra ventaja, es la sencillez y capacidad para monitorizar variables, realizar gráficas, modificar valores de parámetros… Por todo ello Netlogo se trata de una buena plataforma para testar resultados y la evolución del algoritmo.

De forma genérica, y por el posible uso de estos términos en este capítulo, en el escenario de Netlogo existen tres tipos de agentes: tortugas (turtles), links y patches.

Las tortugas son los agentes susceptibles de realizar acciones en el mundo. Podremos programar todo tipo de comportamientos, crear razas diferentes, definir infinitas propiedades atribuibles a esa raza, cambiar su color, tamaño, forma… De alguna forma, son los agentes más activos dentro de la programación.

Los links son enlaces entre tortugas. Serán útiles cuando queramos atribuir algún vínculo entre dos tortugas. También pueden recibir propiedades y se caracterizan por las variables “end1” y “end2” que indican el origen y final del enlace. En nuestro caso, está clara una aplicación directa en los arcos de los grafos.

Por último, los patches son cada una de las celdas que constituyen el mundo. Por defecto el mundo sale teñido de negro, pero nos puede interesar modificar los colores con objeto de mostrar algún escenario particular por pantalla, e incluso su evolución ante un fenómeno.

En este capítulo comentaremos desde la preparación de las librerías de proyectos para ejecutar, las particularidades de la programación o la organización de la misma. Además, incluiremos algunas de las salidas gráficas que conforme se ejecuta el programa podemos ver.

Como aportación en este PFC, se han realizado modificaciones al método original del artículo, en pro de obtener mejores resultados en la resolución del PPSSP e incluso, aportar detalles más realistas que algunos de los considerados por Shou & Huang. Así, concluiremos comentando cuáles

son las diferencias entre ambos métodos e introduciremos el siguiente capítulo que valorará la comparativa de resultados entre ambos modelos.

V.a. ¿Cómo cargar una cartera? Librería de proyectos

El primer paso es ejecutar el problema multiproyecto que queramos evaluar. Al presionar en el botón *Setup* de la aplicación una ventana nos propone cargar uno de los proyectos de la librería. El archivo tiene formato *.xml* como el que se muestra en la siguiente imagen.

```
<?xml version="1.0"?>
<!DOCTYPE mp-list SYSTEM "mp.dtd">
- <mp-list>
  - <mp>
      <name>mp_j03_a4_publi</name>
    - <project-list>
      - <project>
          <filename>KolischInstanzen/j03/j3015_1.sm</filename>
          <start>0</start>
        </project>
      - <project>
          <filename>KolischInstanzen/j03/j3015_2.sm</filename>
          <start>0</start>
        </project>
      - <project>
          <filename>KolischInstanzen/j03/j3015_3.sm</filename>
          <start>0</start>
        </project>
      - <project>
          <filename>KolischInstanzen/j03/j3015_4.sm</filename>
          <start>0</start>
        </project>
      </project-list>
    - <resources>
        <resource>3</resource>
        <resource>0</resource>
        <resource>0</resource>
        <resource>0</resource>
      </resources>
  </mp>
</mp-list>
```

Figura. Fichero *.xml* con las rutas a los proyectos de la cartera

Como se puede observar, con la estructura básica de un documento *.xml* (<datos>…</datos>), en primer lugar aparece el nombre de la cartera "mp_j03_a4_publi". El dígito que acompaña a la letra *a*, nos indica el número de proyectos que contiene esta cartera. En este caso, por tanto, son cuatro, cuyas rutas respecto a la cartera aparecen justo debajo.

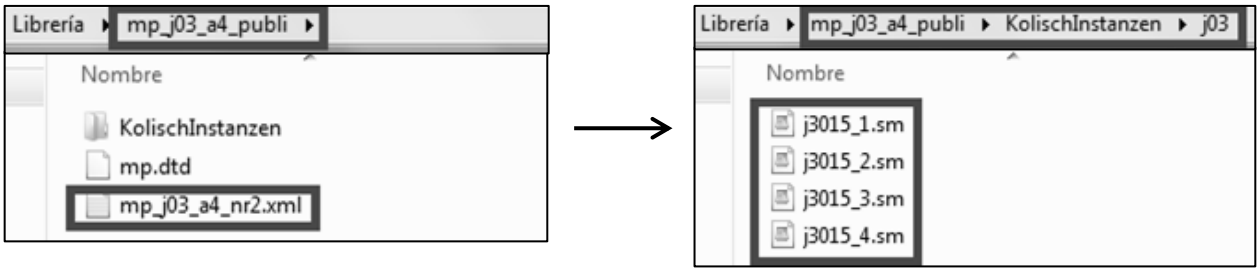
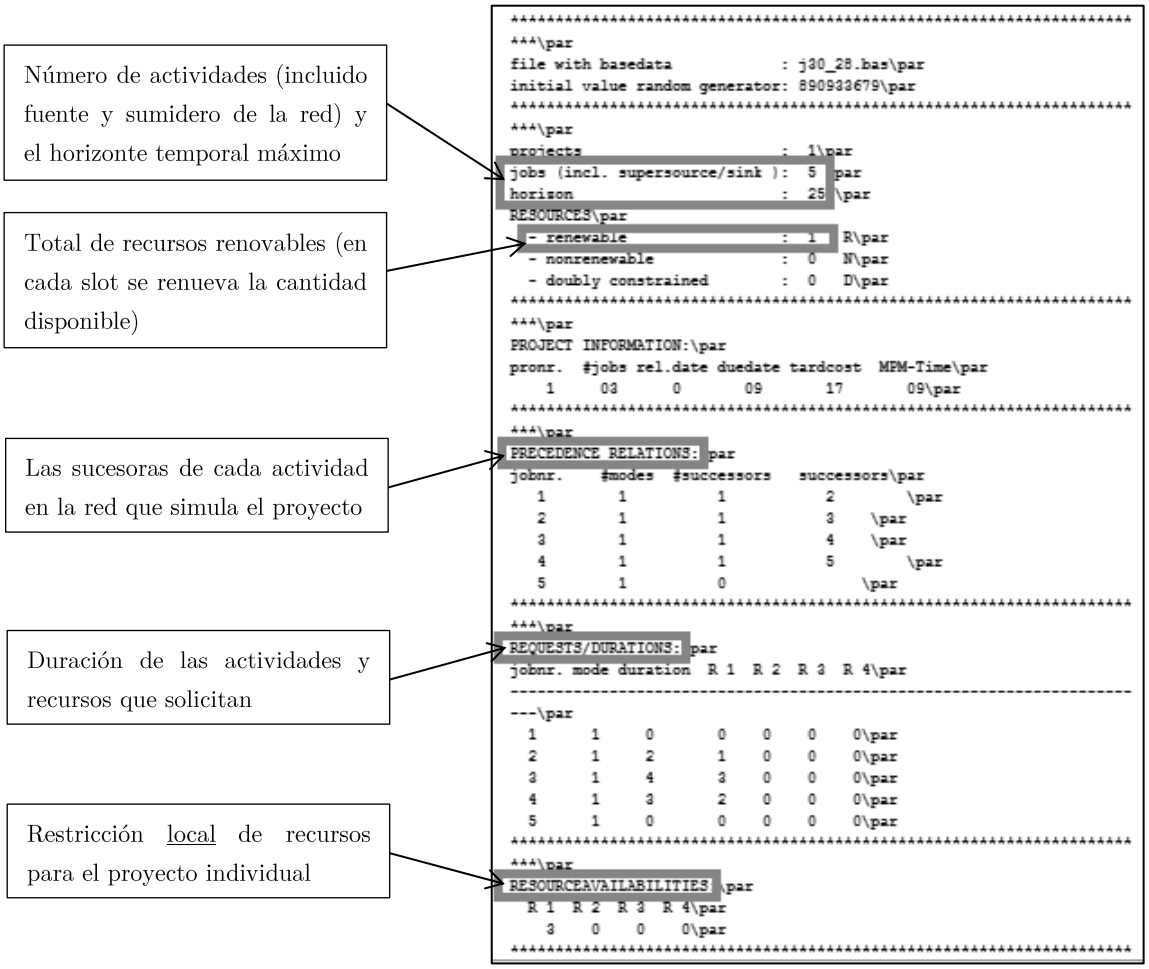


Figura. Rutas de acceso de la cartera de proyectos

Para terminar, el último dato que aparece, es la restricción global de recursos, es decir, las unidades permitidas de consumo en cada slot. De forma genérica podríamos encontrar hasta cuatro recursos diferentes, por lo que nos citan cuatro valores. En este ejemplo, como vemos, sólo hay limitación para el primer recurso, en un total de tres unidades.

Ahora anaizaremos el formato del *.sm* de uno de los proyectos, dando especial relevancia a los datos más importantes para la programación.



Es importante conocer y respetar escrupulosamente el formato de los documentos que hemos visto, puesto que de ellos depende el que la cartera se ejecute de forma correcta en la aplicación Netlogo. En caso contrario, muy probablemente nos daría error al intentar iniciarse el método.

V.b. Análisis de la interfaz. Salidas del programa en Netlogo

Como ya comentamos, una de las principales ventajas de Netlogo era el hecho de que podamos sacar fácilmente por pantalla resultados de la programación. El apoyo visual consigue hacer mucho

más rápida la interpretación de resultados en buena parte de los problemas que podamos encontrar, sean del ámbito que sea.

Si pensamos en nuestro algoritmo, algunas de las variables que más nos interesará monitorizar serían:

- **Límites inferior y superior:** como ya dijimos, son dos valores que conforme transcurran las iteraciones, tendrán que ir aproximándose. Por ello, la representación gráfica de estos nos da pistas de cómo de cerca podemos estar de la programación óptima.

- **Precios en cada ronda:** el funcionamiento del algoritmo gira en torno a los precios de los recursos (la demanda en cada slot de los proyectos con respecto a la disponibilidad de reursos del subastador, hará variar los precios según la ley de oferta y demanda en cada ronda) por lo que, será interesante tener un gráfica que redibuje en cada iteración cuál es el precio de éstos a lo largo del horizonte temporal de la cartera.

- **Mejor solución hasta el momento:** recordamos que consideraremos la mejor solución de la programación a aquella que tenga un mayor límite inferior, ya que por definición, será aquella que mayor beneficio nos estará reportando. Nos sirve para ver la evolución del algoritmo y, en el caso de intuir el resultado del problema cargado, poder confirmar el correcto funcionamiento.

- **Salidas en el "mundo":** denominamos mundo a la pantalla principal de Netlogo donde se puede hacer el seguimiento de las evoluciones de los sistemas. Para nuestro programa, hemos decidido que aparezcan los grafos de cada proyecto, la resolución gráfica del grafo por el método Push-Relabel, la demanda de recursos solicitada por los proyectos con utilidad positiva al programarse individualmente y la solución de la programación por el método serie. Como desventaja frente a estas salidas, tenemos que ralentiza la ejecución considerablemente. Por ello, para el posterior análisis de experimentos donde se cargarán una batalla de carteras, se realizó una versión donde quedan limitadas las salidas visuales.

A continuación mostraremos una serie de imágenes del programa en ejecución que demuestran lo explicado.

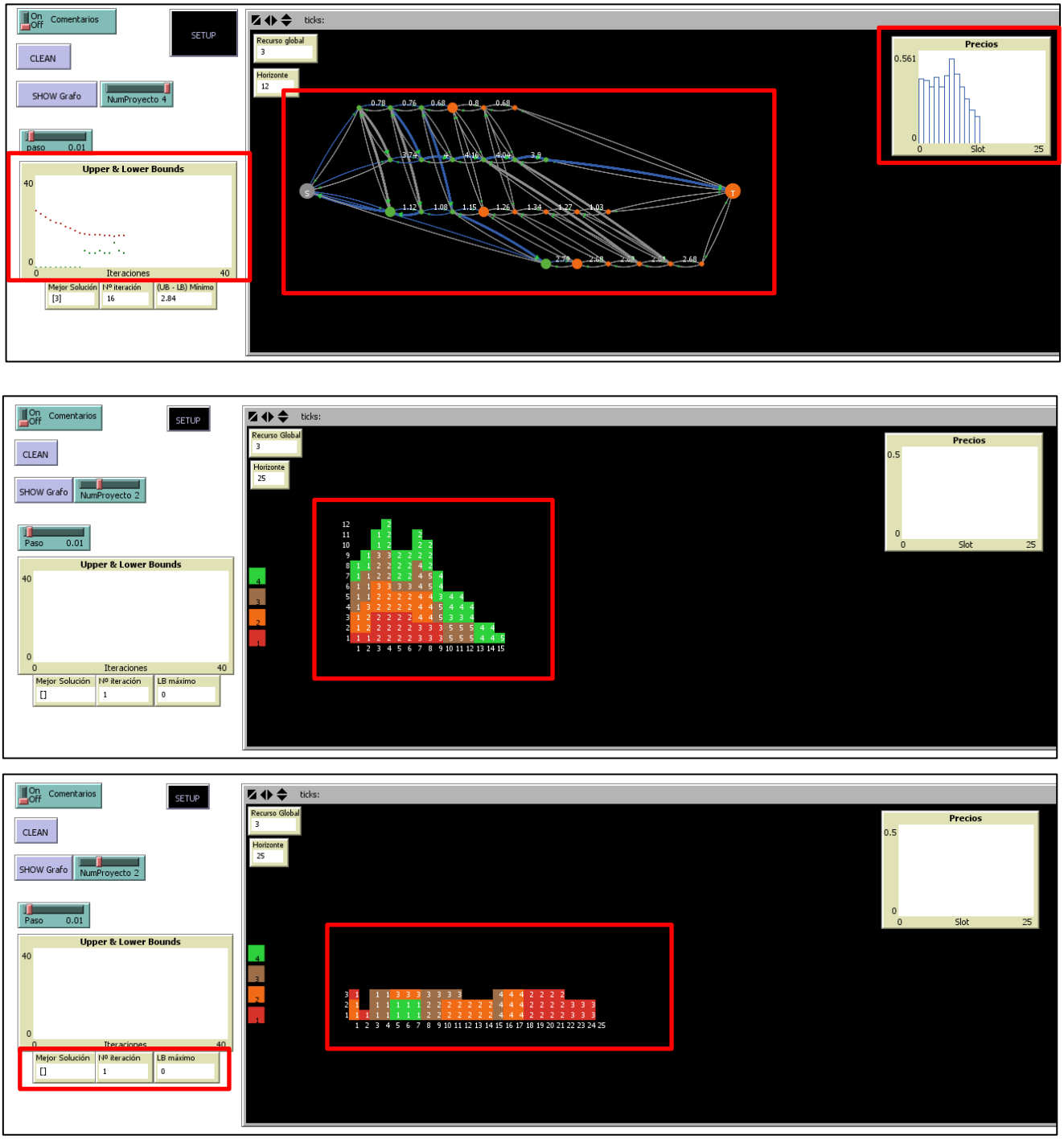


Figura. Salidas de la aplicación durante su ejecución.

V.c. Organización de la programación. Principales procedimientos

Si bien cualquier programación es particular e inherente a la persona que la realiza, en este apartado se pretende explicar de forma genérica cuál es la organización básica en la programación del algoritmo, de tal forma que podamos intuir cuáles son las conexiones en toda ella. Además, aun

siendo todos ellos necesarios, analizaremos los principales procedimientos y cuál es la función que realizan.

Para evitar una ristra de código en una única ventana en Netlogo, lo que dificultaría bastante su entendimiento y manipulación, toda la programación queda dividida en 14 apartados (con formato específico de Netlogo, *.nls*). Quedan incluidos en la cabecera de la programación, de tal forma que conseguimos tener 14 pestañas individuales que gestionan diferentes partes de la misma.

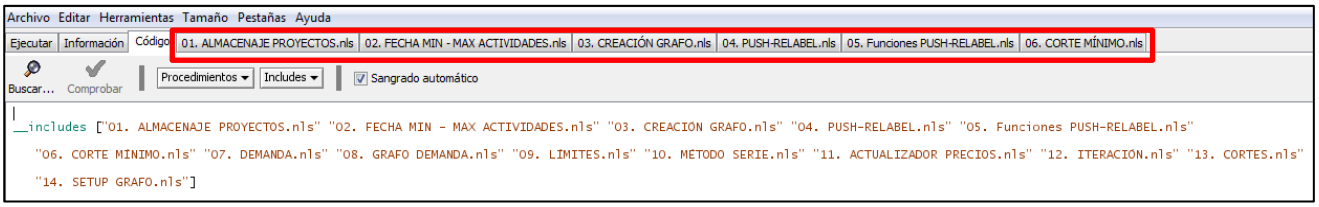


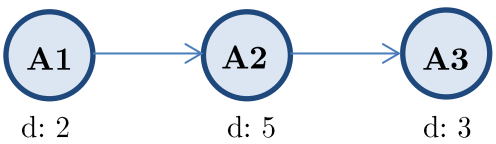
Figura. Cabecera de la programación en Netlogo. Ejemplo de las pestañas individuales.

A continuación comentaremos brevemente cuáles son las funciones que realiza cada fichero:

- **01. ALMACENAJE PROYECTOS:** sólo se ejecuta una vez para cada proyecto de la cartera, de tal forma, que en la carga inicial se almacena toda la información necesaria para cada proyecto en un agente de la raza "proyectos". Algunas de las variables que se almacenan son: número de actividades, actividades sucesoras, posibles cortes del grafo, recursos necesarios para sus actividades….

- **02. FECHA MIN-MAX ACTIVIDADES:** centrándonos sólo en relaciones de precedencia, si programáramos todas las actividades en la fecha más temprana posible, llegaríamos al llamado tiempo de camino crítico (CP, critical path). Por otra parte, al tener un límite en el horizonte temporal, también habrá la fecha más tardía posible de ejecución de dicha actividad teniendo en cuenta las duraciones de todas las actividades. En este apartado se calculan estas dos fechas para todas las actividades. Todos los posibles slots entre la fecha mínima y máxima son los que quedarán representados en su grafo. Las dos fechas quedan en un vector como propiedad del proyecto, por ello también se ejecuta una única veza al inicio.

  Veamos un pequeño ejemplo: un proyecto con tres actividades (A1, A2 y A3) y con duraciones de 2, 5 y 3 unidades de tiempo, respectivamente.



  Teniendo en cuenta las relaciones de precedencia, muy simples en este caso, la actividad A1 podrá empezar desde el instante 0, la actividad A2 desde el instante 2 (0 + 2) y la actividad A3 desde el instante 7 (2 + 5). Finalmente, añadiendo la duración de A3 vemos como el tiempo

de camino crítico es de 10 unidades de tiempo (el proyecto en cuestión no puede finalizarse antes aunque se pretenda).

Si ahora, por ejemplo, consideramos un horizonte temporal máximo de 15 unidades, comenzaríamos en orden inverso al anterior. Con una duración de 3, la actividad A3 podría ser programada hasta el instante 12 (15 – 3), la actividad A2 hasta el instante 7 (12 – 5) y la primera actividad, por lo tanto, hasta el instante 5 (7 – 2). Por tanto, todos los posibles inicios quedarían de la siguiente forma:

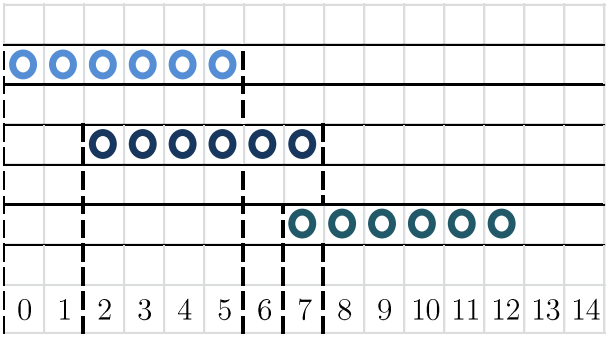
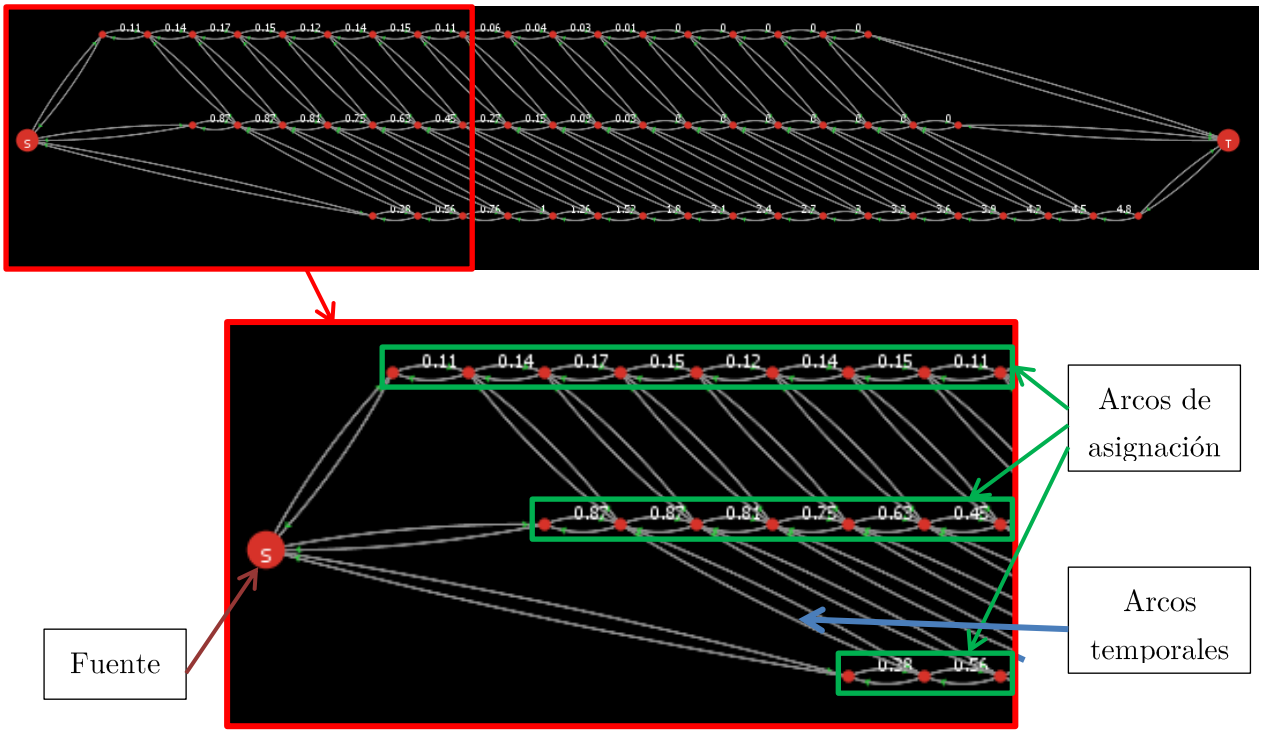


Figura. Ejemplo de posibles inicios de las actividades

- **03. CREACIÓN GRAFO:** gracias al cálculo del anterior procedimiento ya sabemos todos los posibles inicios de cada actividad de un proyecto cualquiera. El grafo se traduce en un número de nodos y arcos que muestran de alguna forma, los caminos admisibles de programación del proyecto. Al final obtenemos la red que el método Push-Relabel tendrá que solucionar en cada iteración. En este procedimiento también se inicializan la capacidad de los arcos en cuestión.



Como vemos en la imagen obtenida desde el programa, los arcos son siempre de doble sentido entre los nodos. El número que aparece sobre los arcos de asignación es la capacidad del arco en esa iteración.

- **04. PUSH-RELABEL / 05. Funciones PUSH-RELABEL:** básicos en la programación, son dos de los procedimientos más importantes y más complejos del ejecutable. El primero hace llamada al método y se encarga de preparar el grafo para la nueva resolución. En el segundo, ya está implementado propiamente el algoritmo con las funciones de Push y Relabel ya explicadas en anteriores capítulos. Gráficamente, se va viendo a cada instante qué conjunto de nodos están activos (con exceso de flujo) y qué arcos están admisibles (no están saturados) lo que hace que la representación sea bastante didáctica.

- **06. CORTE MÍNIMO:** una vez que se sabe qué flujo máximo ha alcanzado el sumidero de la red, y en virtud del famoso teorema de Ford-Fulkerson (1962) (*"En cualquier red, el flujo máximo que fluye de la fuente al destino es igual a la capacidad del corte mínimo que separa a la fuente del destino"*), procedemos a calcular el corte mínimo, que nos indicará el inicio de la programación de las actividades más económico (irá variando en cada iteración conforme se modifiquen los precios según la oferta y demanda). La ventaja que nos da tener el valor del flujo máximo, es reducir la búsqueda entre todos los posibles cortes, ya que en el momento que un corte tenga ese valor, habremos llegado a la solución. Esto supone un ahorro computacional importante y más en redes de gran tamaño.

- **07. DEMANDA / 08. GRAFO DEMANDA:** si con el corte mínimo hemos obtenido la programación que más interesa a cada proyecto por ser la más económica, podremos saber en el conjunto de la cartera, qué demanda de recursos se tiene en cada slot temporal. Evidentemente, es muy probable que la demanda de todos los proyectos supere a la restricción de recurso global que permitimos en cada slot, luego de momento, nos encontramos con una programación no factible.

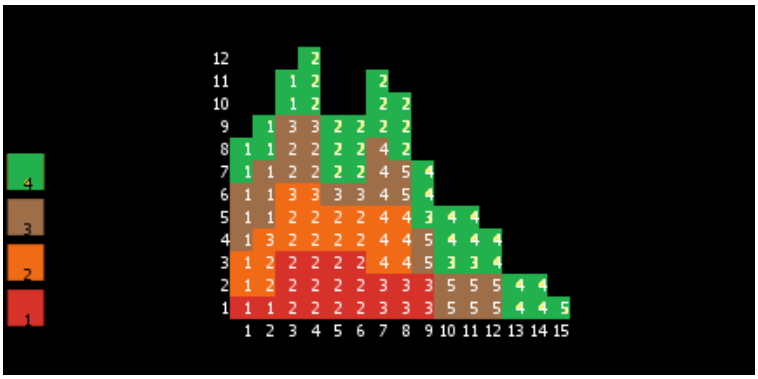


Figura. Ejemplo de la demanda global de los cuatro proyectos según su interés

Este cálculo es importante, ya que la actualización de precios para la siguiente ronda tendrá en cuenta qué slots son los más saturados para incrementar así su precio, de tal forma que la subasta vaya consiguiendo que a algunos proyectos comiencen a no compensarles la programación en esa zona y se vean desplazados a zonas de menos demanda.

- **09. LÍMITES:** recoge las funciones de cálculo del límite superior e inferior de la solución. Por tanto, será ejecutado en dos instantes diferentes: tras tener la subcartera con proyectos de utilidad positiva (límite superior, UB) y tras ejecutar el método serie y ver qué proyectos concluyen y en qué instante respecto a su tiempo de camino crítico.

- **10. MÉTODO SERIE:** como ya hemos dicho, hasta este momento tenemos una demanda de los proyectos minimizando costes, pero no se ha tenido en cuenta la disponibilidad de recurso en cada instante. El objetivo del método serie era dar a cada una de las actividades un índice de prioridad según el cual se van a ir programando, siempre y cuando sea posible. Esta función es bastante compleja puesto que la casuística a la hora de insertar nuevas actividades y los slots que se van ocupando, es bastante variada. En la imagen vemos un ejemplo en el que, el recurso global permitido es de tres unidades y el horizonte temporal máximo es de 25 slots.

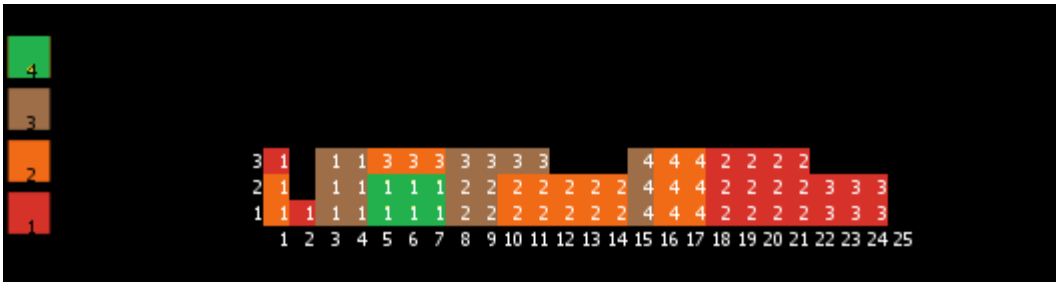


Figura. Secuenciación de actividades según el método serie

El índice de prioridad para una actividad se define como el inicio en el que querría ser programada más la mitad de su duración. De esta forma, se van incluyendo de menor a mayor índice (según la definición, estarán en la cabecera aquellas actividades que quieren programarse cuanto antes y que además son cortas).
Como mejora introducida respecto al método serie estándar, además del orden según el índice comentado, se desarrolló una heurística que tuviera en cuenta cuántas actividades de cada proyecto estaban ya programadas, para así intentar que los proyectos puedan terminar cuanto antes.

- **11. ACTUALIZADOR DE PRECIOS:** al final de la consecución de la iteración se actualizan los precios de los arcos de asignación en función de la demanda de recursos que

hubiera habido en los slots. La actualización depende también de un paso s, el cual puede ser fijo o bien variable. Ambas alternativas son contempladas en la publicación.

- **12. ITERACIÓN:** a diferencia de la primera iteración, la cual hace uso de más procedimientos con objeto de inicializar el algoritmo, el resto se ven automatizadas con este apartado de la programación. Para valorar el funcionamiento del algoritmo, un factor importante es la determinación del criterio de parada. En nuestro caso, se terminará la ejecución en el caso de que ningún proyecto tuviera utilidad positiva (la mejor programación del proyecto no está compensada por el valor que aporta) o bien, no se haya encontrado una mejor solución (es decir, un LB mayor) tras cuarenta iteraciones.

- **13. CORTES:** en nuestro caso particular, las redes con las que tratamos tienen menos cortes de los que podríamos considerar en primer término. Teniendo en cuenta que la capacidad de los arcos temporales es infinita, para que el flujo por ellos nunca se vea limitado, la búsqueda del corte mínimo sólo va a poder incluir un arco de asignación de cada actividad. Todas las combinaciones posibles de estos arcos que respetan las restricciones de precedencia, son almacenadas inicialmente como una variable más del proyecto. Como las opciones de cortes siempre van a ser las mismas para el proyecto, es más útil para la búsqueda en cada iteración.

- **14. SETUP GRAFO:** es un procedimiento que reinicia el grafo según estaba antes del proceso de Push-Relabel. Es necesario para hacer cálculos con las capacidades iniciales en los arcos.

Como hemos visto, los diferentes bloques de la programación están divididos de una forma lógica al tener presente siempre el funcionamiento global del algoritmo.

A continuación, mostramos en un diagrama las conexiones entre las funciones de la programación. Aquellos que no están sombreados, sólo se realizan en la primera iteración, tantas veces como proyectos tenga la cartera. En esta primera ocasión, no es necesario ejecutar el algoritmo de Push-Relabel y corte mínimo, ya que los precios en la subasta son cero, luego la capacidad de los arcos es nula y no hay flujo a través del grafo. Esto supone que la programación deseada por cada proyecto es la correspondiente al tiempo de camino crítico (como ya sabemos la solución inicial, directamente se pasa a la función *schedule,* se calcula la demanda global y se actualizan precios).

El resto de iteraciones ya son iguales y se ejecutan en las funciones sombreadas, desde el procedimiento *iterar*.

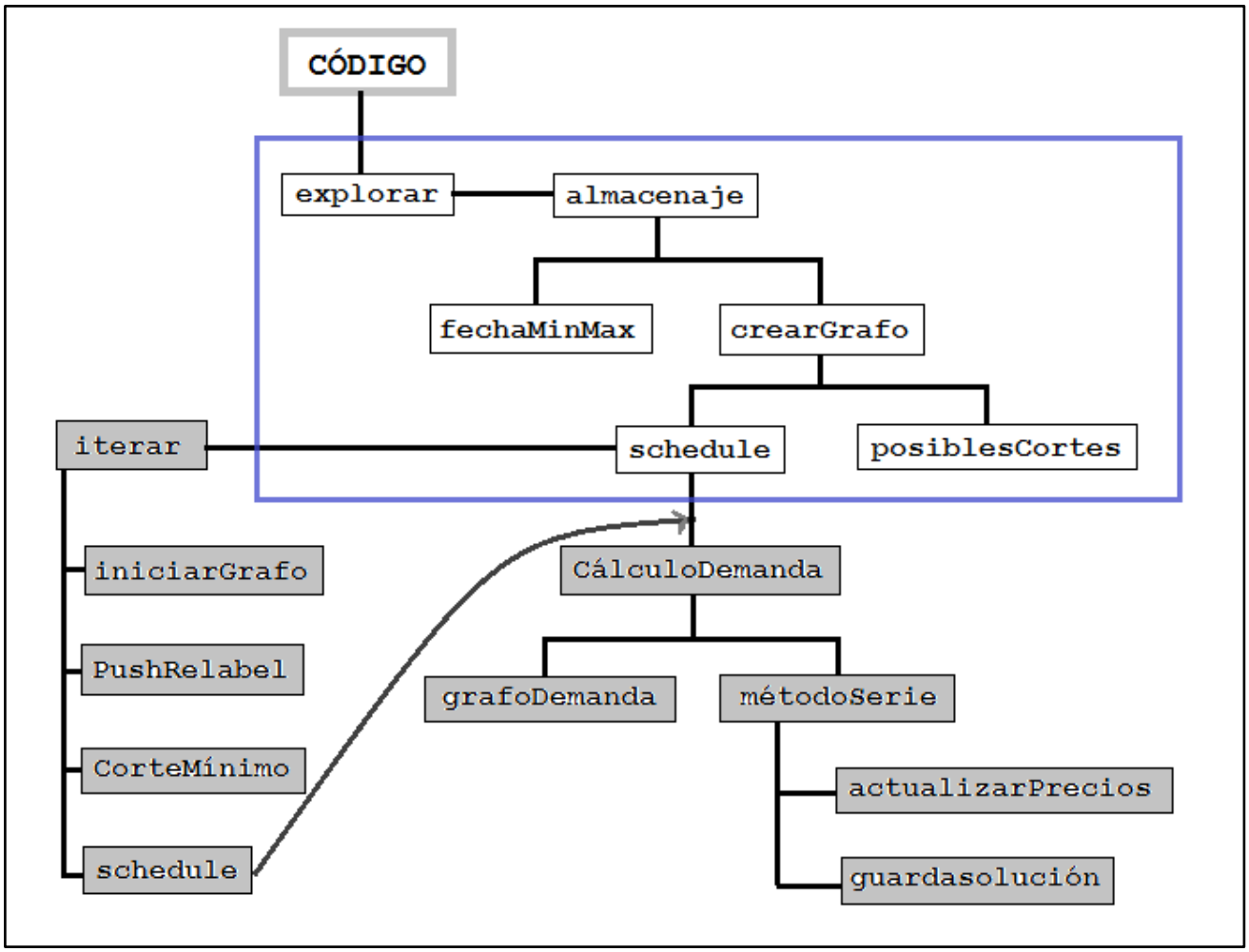


Figura. Grafo de conexiones en la programación

V.d. Diferencias entre los dos métodos propuestos

Todo lo visto hasta el momento en este capítulo son características genéricas de la estructuración de la programación, que son idénticas en ambos métodos, pero, ¿qué debilidades encontramos en el método original y que fueron susceptibles de mejora?.

Uno de los detalles más llamativos en el artículo es la forma de valoración de los proyectos. Como ya comentamos, su decisión (a lo que denominaban NPV) era que el valor máximo del proyecto se viera reducido en un 5% por cada unidad de tiempo que la finalización del proyecto se viera alejada de su tiempo de camino crítico.

Nuestra primera crítica puede ir ya encaminada en ese sentido: ese criterio de valoración no es lo que por todos está conocido como el NPV o VAN. Necesariamente habría que incluir una tasa de descuento k que modifique de forma realista ese único *inflow* que tenemos cuando el proyecto finaliza.

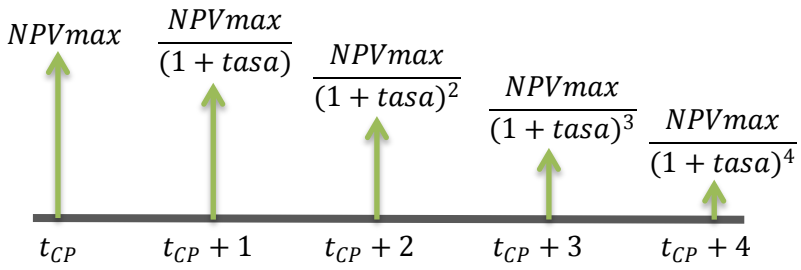


Figura. Método de valoración de proyectos incluido en la aplicación

Como vemos, conforme el proyecto se retrasa respecto del tiempo de camino crítico (t_{CP}, que recordamos que es el mínimo tiempo necesario para ejecutarse un proyecto, simplemente respetando relaciones de precedencia), su valor, evidentemente, se va viendo disminuido, pero no en un porcentaje cualquiera que se quiera estipular. Planteamos un ejemplo sencillo en el que queda presente la diferencia entre considerar la valoración del artículo y ésta.

Imaginemos un proyecto que tiene como valor máximo 10 unidades. Si concluye 5 slots más tarde respecto de su t_{CP}, ¿cuánto reportaría el proyecto según las dos valoraciones? Consideremos una tasa de descuento k del 5%.

VALORACIÓN DE PROYECTOS		EJEMPLO
Según artículo	$NPVmáx \times 0.95^{(t-t_{CP})}$	$10 \times 0.95^5 = 7.738$
Método propuesto	$\frac{NPVmáx}{(1+k)^{(t-t_{CP})}}$	$\frac{10}{1.05^5} = 7.835$

Como vemos, la diferencia no es tanto en valor (aunque si consideramos un valor k más realista, de un 10%, ya hablamos de 6.209 frente a 7.738), si no conceptualmente. Sabemos que el tratamiento correcto de los flujos de caja en el tiempo, es éste.

Otro punto clave del que ya hicimos comentarios previamente, es la influencia de la heurística de asignación de prioridades. Recordamos como el método utilizado era el método serie, pero ¿cuál es la ley de prioridad?: ¿primero las actividades más cortas?, ¿las que menos recursos consumen?, ¿las que pertenezcan a los proyectos con mayor beneficio?

Si bien, sabemos que el criterio que decida tomar el Project manager es muy importante, tampoco sabemos con certeza cuál es la mejor receta de forma inequívoca. Quizás, esa combinación de criterios o incluso la variación de éstos a lo largo de las iteraciones sea la clave del éxito.

Inicialmente, la regla que estaba considerada era sumar al tiempo de inicio de la actividad si pudiera programarse libremente según su interés (sin problemas aún por superar la cantidad de recurso global o no) la mitad de la duración de dicha actividad. Definida así la prioridad, el mínimo

valor numérico supone tratar con la actividad que más interés tiene en programarse al inicio del proyecto y de más corta duración. Luego, se ordenaban las actividades de menor a mayor "coeficiente" y así se iban cuadrando en el método serie hasta que fuera posible. Cuando se superara la fecha máxima, observaríamos qué proyectos han conseguido finalizar y cuáles quedaron fuera.

De entre todas las posibles opciones y variantes que el cambio del criterio de prioridad nos proporciona, decidimos pensar en alguna que tenga en cuenta cuánto beneficio supone que se ejecute ese proyecto, por eso introducimos el valor de $NPV_{máx}$. De esta forma, estaremos diciendo que una actividad es más importante si pertenece a un proyecto más rentable. Por ello, la nueva regla de prioridad sería ese inicio deseado por la actividad si en su mano estuviera, dividido por el valor máximo del proyecto. En este caso, el menor número supone una actividad que quiere programarse pronto y que además nos conviene, puesto que finalizar ese proyecto reportará más beneficio.

CRITERIO DE PRIORIDAD MÉTODO SERIE

Según artículo	$inicio + 0.5 \times duración$
Método propuesto	$\frac{inicio}{NPVmáx}$

Como vemos, no hay ninguna relación entre los dos criterios, luego la comparativa en la batalla de experimentos que analizaremos en el próximo capítulo será muy interesante, puesto que de primeras no podemos posicionarnos en cuál dará mejores soluciones.

Por otra parte, en ambos casos, partiendo de la primera ordenación de las actividades según los criterios, también añadimos otras dinámicas de refinamiento, como son los siguientes:

- Durante dos iteraciones, recorremos la lista de tal forma que si una actividad usa menos recursos que la anterior, pueda adelantar su posición. Así, podemos priorizar aquellas actividades que demandan menos recursos, y que quizás puedan encajar más fácilmente en el puzle que, en definitiva, es la programación.

- Mediante contadores vamos controlando, conforme avanzamos en la lista, el número de actividades de cada proyecto que han ido apareciendo. Con esta información, priorizamos una actividad respecto a su anterior si tiene más actividades de su proyecto programadas hasta el momento que la otra. De esta manera, conseguimos fomentar la finalización de los proyectos cuanto antes. Para esta mejora, recorremos la lista un total de 3 veces.

Otro factor a tener en cuenta, y muy importante en la obtención de la mejor solución, es el criterio de parada. En la literatura, evidentemente hay muchas opciones, desde el más sencillo que

es un número de iteraciones fijo, hasta teniendo en cuenta la aproximación de los límites superior e inferior de la solución.

Netlogo presenta una gran ventaja como soporte visual, pero bien es cierto, que su programación por capas hace que las resoluciones se vean ralentizadas en comparación a otros lenguajes de programación (y también, esas salidas gráficas tienen parte de responsabilidad). Por ello, incluir un criterio en relación a los límites podía ser algo incómodo a la hora de que, incluso pequeñas carteras (como las que tratamos) pudieran necesitar demasiado tiempo de ejecución. Por otra parte, tampoco se puede prever qué gap va a haber en cada caso entre los límites cuando se llegue a la solución óptima.

Para solventar este problema, la decisión que se tomó es que, si durante 40 iteraciones no se ha encontrado una mejor solución respecto a la que tenemos, el programa parará. Si por el contrario se encuentra una mejora, el contador empezará desde el principio. Consideramos que es un número idóneo como para confiar en que el tiempo de desarrollo que se da al algoritmo sea suficiente. En todo caso, al tratar todos los problemas bajo este mismo criterio, todos tendrán la ventaja o desventaja que haya supuesto esta decisión.

Por último, según como definimos el funcionamiento del algoritmo, la solución óptima se obtiene cuando hay un equilibrio en los precios de los recursos a lo largo del horizonte del proyecto. La actualización de precios y el paso también es crítico. Por ello, además de considerar posibles pasos fijos, se optó por la programación con un paso variable, de tal forma que, si hay más demanda en un slot de la admisible, subimos los precios con un paso, pero si sobran recursos, reducimos los precios con un paso distinto. En nuestro caso concreto, tomamos un paso de subida y la mitad para el de bajada. La conveniencia o no de este criterio, también se podrá comprobar en el análisis del próximo capítulo.

V.e. Soluciones para el problema del artículo

Como ya anticipamos en el capítulo anterior, la solución que encontraba el artículo, era la siguiente:

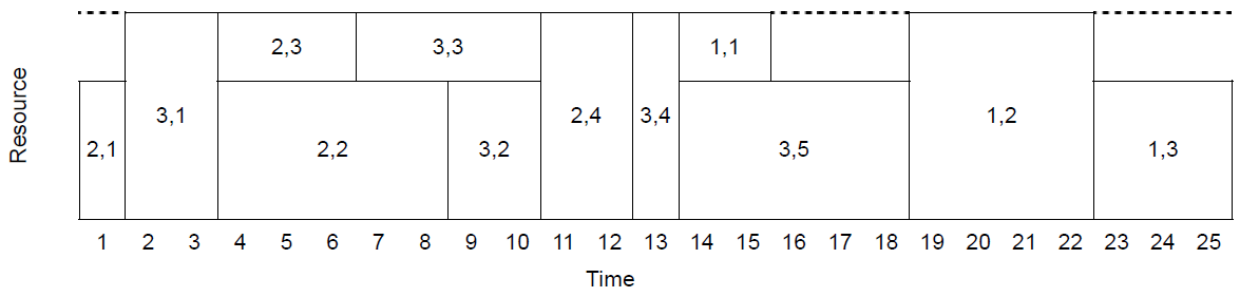


Figura. Solución de la programación para el problema del artículo

Según el método de valoración que ya incluye la tasa de descuento ($k = 0.05$) para los flujos de caja, podemos ver cuánto beneficio reporta esta programación. Recordamos como el tiempo máximo de horizonte temporal es de 25 unidades. Al conocer los proyectos, sabemos su tiempo crítico, y dependiendo de cuánto se alejen de éste en la programación solución, el descuento del valor máximo del proyecto será mayor o menor. En la siguiente tabla desglosamos el cálculo en cada caso. La subcartera solución es $\{2,\ 3,\ 1\}$ según el orden de finalización.

Proyectos finalizados	t_{CP}	Tiempo finalización	$NPV_{máx}$	Valor del proyecto
Proyecto 2	8	12	8	$\frac{8}{(1+0.05)^{(12-8)}} = 6.582$
Proyecto 3	12	18	11	$\frac{11}{(1+0.05)^{(18-12)}} = 8.208$
Proyecto 1	9	25	6	$\frac{6}{(1+0.05)^{(25-9)}} = 2.749$
			TOTAL	**17.539**

Pues bien, a continuación presentamos las soluciones que nuestra programación nos da. Evidentemente, el método I debía conseguir la misma solución que la original.

En la iteración número 36 conseguimos la siguiente programación:

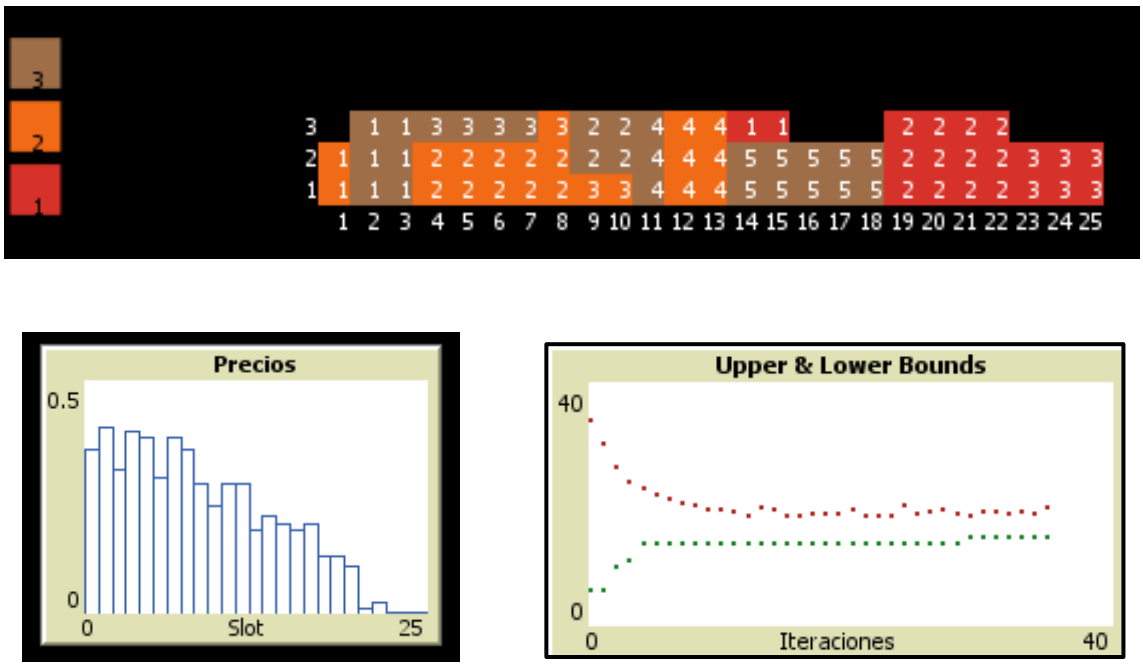


Figura. Salidas para la programación solución en la aplicación de Netlogo (método I)

Como vemos, la programación es idéntica excepto en que la actividad 4 del proyecto 3 se intercambia con la actividad 4 del proyecto 2, lo que hace que el tiempo de finalización del proyecto 2 sea en el slot 13 en vez del 12. A pesar de que no supone una gran diferencia ni en el valor de la cartera solución como comprobaremos ahora, cabe justificarlo debido a alguno de los métodos introducidos además del criterio de prioridad que hayan prevalecido una actividad respecto a la otra. Si hacemos un análisis similar al anterior:

Proyectos finalizados	t_{CP}	Tiempo finalización	$NPV_{máx}$	Valor del proyecto
Proyecto 2	8	13	8	$\frac{8}{(1+0.05)^{(13-8)}} = 6.268$
Proyecto 3	12	18	11	$\frac{11}{(1+0.05)^{(18-12)}} = 8.208$
Proyecto 1	9	25	6	$\frac{6}{(1+0.05)^{(25-9)}} = 2.749$
			TOTAL	**17.225**

Al ejecutar con el método II el problema, volvemos a encontrar la subcartera solución de $\{2,\ 3,\ 1\}$, pero con una programación sustancialmente diferente:

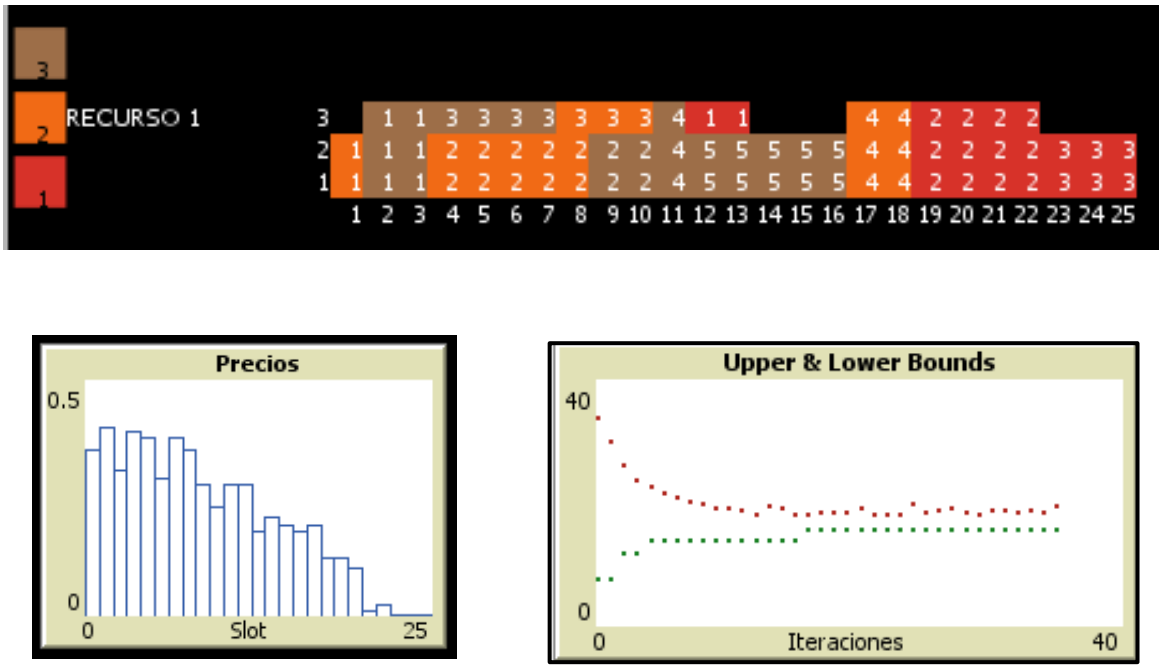


Figura. Salidas para la programación solución en la aplicación de Netlogo (método II)

Proyectos finalizados	t_{CP}	Tiempo finalización	$NPV_{máx}$	Valor del proyecto
Proyecto 3	12	16	11	$\frac{11}{(1+0.05)^{(16-12)}} = 9.050$
Proyecto 2	8	18	8	$\frac{8}{(1+0.05)^{(18-8)}} = 4.911$
Proyecto 1	9	25	6	$\frac{6}{(1+0.05)^{(25-9)}} = 2.749$
			TOTAL	**16.710**

En este caso, se prioriza la finalización del proyecto 3 frente al proyecto 2. Teniendo en cuenta la regla de prioridad del método II, es lógico, puesto que la finalización del proyecto 3 reporta un mayor valor. El inconveniente es que no se ve compensado por el retraso que supone para el proyecto 2, y por ello como vemos en el cálculo de la tabla, el valor de la cartera global es menor que en el método I.

VI. ANÁLISIS DEL ALGORITMO

Una vez realizada la réplica del algoritmo y la nueva versión que pretende traer consigo mejores soluciones, el siguiente paso sería el análisis de resultados al hacer una batalla con diferentes carteras. Con todo ello, conseguiremos un análisis más pormenorizado de su funcionamiento, su respuesta ante la variación de determinados parámetros del modelo o ver su efectividad (ya que, teniendo carteras pequeñas y proyectos sencillos como es nuestro caso, hasta podemos plantearnos la solución a mano y ver si realmente se está llegando a la solución óptima).

Los cuatro proyectos que hemos utilizado son los que habíamos presentado en el capítulo IV. Somos conscientes de que no llega a ser del todo representativo el tener proyectos tan pequeños, pero también queremos tener carteras abordables para la densa programación que hay detrás (el grafo de nodos y arcos para un proyecto de 50 actividades y su resolución con este soporte, sería algo inviable).

A continuación explicaremos la recogida y toma de datos, qué parámetros hemos ido variando y qué resultados obtuvimos con interés para este PFC.

VI.a. Descripción del experimento

Es importante el planteamiento inicial que se quiera dar al experimento y qué datos se pretenden recoger, analizar, puesto que en buena parte, condicionará el éxito del análisis. Evidentemente, dentro de un criterio, cuanto mayor número de parámetros y registros tengamos en cuenta, más completo será el análisis y mejores conclusiones podremos obtener.

Al disponer de cuatro proyectos, las dimensiones de las carteras que podemos configurar son de dos, tres y los cuatro proyectos. Teniendo en cuenta todas las combinaciones diferentes estamos hablando de once carteras diferentes (resaltamos como, en nuestro análisis en particular, será lo mismo ejecutar el proyecto 3 y 4, que el 4 y el 3, luego las combinaciones posibles se ven por ello reducidas).

En nuestro caso en cuestión, podemos plantearnos por una parte, qué datos podemos obtener gracias a ejecutar la aplicación y cuáles simplemente conociendo las características de cada proyecto (duración, recursos, precedencias).

La salida de nuestro programa es la subcartera que se termina ejecutando y el valor que reportará a la organización (que como ya vimos, coincide con el LB del algoritmo en dicha iteración). Si bien es cierto, como es habitual en la literatura, nos podríamos plantear el tiempo de ejecución, o el número de iteraciones en el que se obtiene la mejor solución, pero como ya hemos

comentado, nuestra aplicación por sus características no es competente en cuanto al tiempo de ejecución, y la convergencia de antes o después a la solución depende directamente de la cartera con la que tratemos, luego no es comparable entre casos diferentes.

Por ello, con respecto al grupo de datos obtenidos de la aplicación, nos quedamos con los proyectos elegidos dentro de toda la cartera, sus tiempos de finalización en la programación solución y el valor que reportan, que por otra parte, no deja de ser lo más relevante. Además, hemos creado un nuevo parámetro que nos muestra la media de consumo de recursos en la cartera solución, y que nos va a servir de comparativa en los resultados dados por los dos métodos.

Con respecto al grupo de datos que inicialmente podemos conocer de cada cartera con la que tratemos, procedemos a hacer una descripción de los más interesantes y que han sido tenidos en cuenta. Para ejemplificarlo, iremos calculando cada parámetro para el proyecto número 4, el más complejo de los que tenemos.

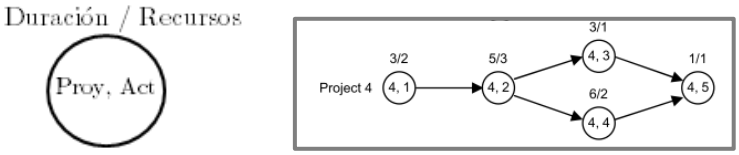


Figura. Proyecto ejemplo sobre el que calcularemos sus parámetros

Resumimos los parámetros principales que hemos tenido en cuenta:

Parámetros en el experimento

- **Número de actividades** (j): como vemos en el grafo, 5 nodos representan las 5 actividades.
- **Duración del camino crítico** (*CPD: critical path duration*): es la fecha más temprana en la cuál el proyecto puede estar finalizado. Teniendo en cuenta las duraciones de las actividades, vemos en la imagen como el tiempo es de 15 unidades.

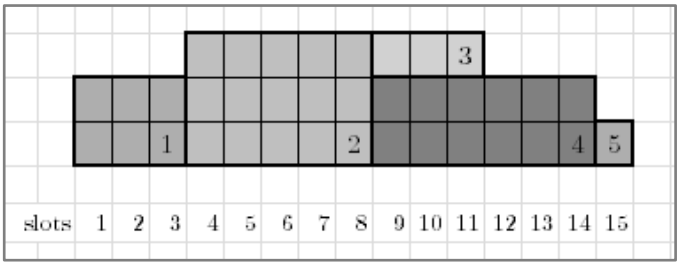


Figura. Programación según camino crítico del proyecto ejemplo

- **Recurso utilizado** (*RU: resource units*): el número de unidades de recurso necesarias para la ejecución completa. En el ejemplo:

$$2 \times 3 + 3 \times 5 + 1 \times 3 + 2 \times 6 + 1 \times 1 = 37\ unidades$$

- ***Rentabilidad*** **del proyecto**: se define como el cociente entre el $NPV_{máx}$ y el RU, es decir, el máximo beneficio que nos reportaría, con respecto al recurso que debemos poner en juego. En principio, si la programación final es próxima a la de su camino crítico (el valor máximo del proyecto será próximo a su $NPV_{máx}$), podemos decir que valores altos de este cociente nos está mostrando proyectos más rentables (mayor beneficio respecto a los recursos consumidos). Si por ejemplo, el $NPV_{máx}$ que nos aporta es de 9 unidades, el ratio es:

$$\frac{NPV_{máx}}{RU} = \frac{9}{37} = 0.243$$

Es un parámetro sencillo y recurrente a la hora de comparar previamente proyectos.

- **Factor medio de utilización** (*AUF: average utilization factor*): para poder definirlo tenemos que saber cuál es la limitación de recurso global. Es un ratio que nos dice cuánto recurso utiliza el proyecto respecto a todo el que podría haber usado hasta el instante en el que concluye. Es decir, imaginemos que el proyecto, al programarse con otras actividades de la cartera, finaliza en el slot 20, y que el recurso global permitido era de 5 unidades por slot, entonces:

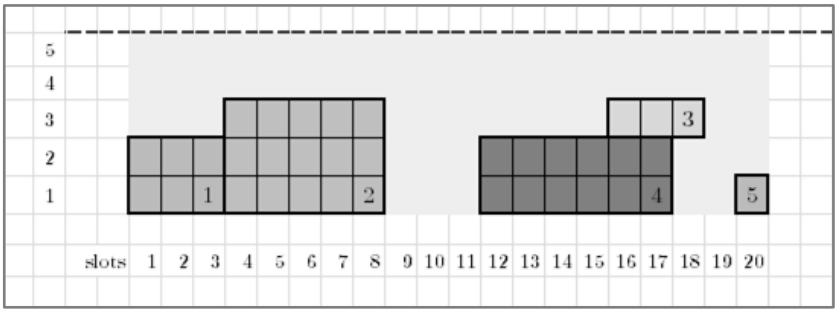


Figura. Posible programación del proyecto ejemplo

Como vemos, frente a las 37 unidades que necesita el proyecto para programarse, nosotros le hubiéramos permitido disponer de $5 \times 20 = 100\ unidades$. Luego, el *AUF* para este proyecto es 0.37. Bien es cierto, que el sentido de este parámetro está propiamente ligado a saber cuando finaliza el proyecto en la solución. Por ello, en principio, si en el cálculo dividiéramos entre los recursos disponibles hasta su tiempo de camino crítico, hablaríamos de un parámetro similar en concepto, pero que no es propiamente el *AUF* utilizado en la literatura.

- **Complejidad de la red:** hay muchas opciones de medida de complejidad de la red, en función de los nodos, los enlaces, las precedencias… En principio, el que un proyecto sea de mayor complejidad podría condicionar el que se termine incluyendo en la subcartera solución. Como en nuestro caso tratamos con proyectos muy simples, no se va a tratar de un valor condicionante en el análisis.

- **Tiempos de finalización:** es lo que define directamente la solución. Sabemos qué proyectos son los elegidos y cuándo se ven concluídos, luego enseguida se puede calcular el valor que reporta esa cartera.

- **AUF medio:** este es un cálculo posterior al conocimiento de la subcartera solución. El objetivo de crear este parámetro es ver rápidamente qué uso se está haciendo de los recursos globales que disponemos.

$$AUF_{medio} = \frac{AUF_{n1} + AUF_{n2} + \cdots + AUF_{nT}}{nT}$$

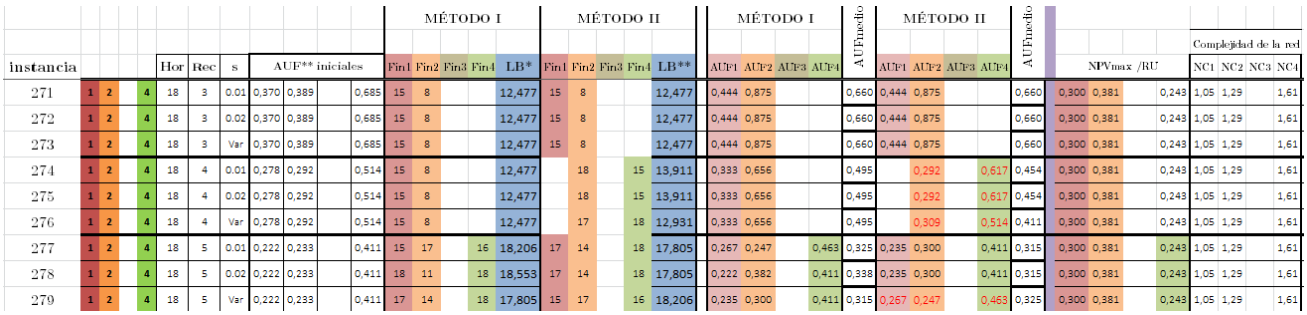
  Como vemos, del número total de proyectos (nT) que conforman la solución, calculamos simplemente su media. El cálculo de este ratio para dos soluciones diferentes, puede darnos conclusiones en relación a la economidad de la solución. Evidentemente, la comparativa se debe hacer al resolver la misma cartera inicial y con la misma restricción global, y aún tiene más sentido en el caso de que los dos métodos incluyan en su solución final el mismo conjunto de proyectos, pero con diferente disposición en la programación (que nos estén dando diferentes tiempos de finalización para los proyectos). En este caso, directamente, podremos decir que una solución es mejor que otra en cuanto a la optimización del uso de recursos (aunque cierto es que no por ello el valor de esa solución tenga que ser mejor).

Recogemos en una tabla resumen todos estos parámetros, dependiendo de si su cálculo se obtiene desde los datos iniciales de los proyectos, o bien tras la ejecución de la aplicación:

Parámetros iniciales	**Resultados de la aplicación Netlogo**
- Número de actividades	- Subcartera solución
- Tiempo de camino crítico (t_{CP})	- Tiempos de finalización de los proyectos (programación)
- Recurso utilizado (RU)	- Lower Bound (LB): valor que reporta la subcartera solución
- Cociente $NPV_{máx}$/RU	- AUF_i
- Complejidad de la red	- AUF_{medio}

Tabla. Resumen de los parámetros valorados en el análisis

Después de haber introducido el significado de los parámetros, podemos ver a través de un ejemplo en nuestra tabla de análisis, cómo se ha ido resolviendo cada instancia.



												MÉTODO I					MÉTODO II					MÉTODO I					MÉTODO II								Complejidad de la red			
instancia				Hor	Rec	s	AUF** iniciales					Fin1	Fin2	Fin3	Fin4	LB*	Fin1	Fin2	Fin3	Fin4	LB**	AUF1	AUF2	AUF3	AUF4	AUFmedio	AUF1	AUF2	AUF3	AUF4	AUFmedio	NPVmax /RU			NC1	NC2	NC3	NC4
271	1	2	4	18	3	0.01	0,370	0,389		0,685		15	8			12,477	15	8			12,477	0,444	0,875			0,660	0,444	0,875			0,660	0,300	0,381	0,243	1,05	1,29	1,61
272	1	2	4	18	3	0.02	0,370	0,389		0,685		15	8			12,477	15	8			12,477	0,444	0,875			0,660	0,444	0,875			0,660	0,300	0,381	0,243	1,05	1,29	1,61
273	1	2	4	18	3	Var	0,370	0,389		0,685		15	8			12,477	15	8			12,477	0,444	0,875			0,660	0,444	0,875			0,660	0,300	0,381	0,243	1,05	1,29	1,61
274	1	2	4	18	4	0.01	0,278	0,292		0,514		15	8			12,477		18		15	13,911	0,333	0,656			0,495		0,292		0,617	0,454	0,300	0,381	0,243	1,05	1,29	1,61
275	1	2	4	18	4	0.02	0,278	0,292		0,514		15	8			12,477		18		15	13,911	0,333	0,656			0,495		0,292		0,617	0,454	0,300	0,381	0,243	1,05	1,29	1,61
276	1	2	4	18	4	Var	0,278	0,292		0,514		15	8			12,477		17		18	12,931	0,333	0,656			0,495		0,309		0,514	0,411	0,300	0,381	0,243	1,05	1,29	1,61
277	1	2	4	18	5	0.01	0,222	0,233		0,411		15	17		16	18,206	17	14		18	17,805	0,267	0,247		0,463	0,325	0,235	0,300		0,411	0,315	0,300	0,381	0,243	1,05	1,29	1,61
278	1	2	4	18	5	0.02	0,222	0,233		0,411		18	11		18	18,553	17	14		18	17,805	0,222	0,382		0,411	0,338	0,235	0,300		0,411	0,315	0,300	0,381	0,243	1,05	1,29	1,61
279	1	2	4	18	5	Var	0,222	0,233		0,411		17	14		18	17,805	15	17		16	18,206	0,235	0,300		0,411	0,315	0,267	0,247		0,463	0,325	0,300	0,381	0,243	1,05	1,29	1,61

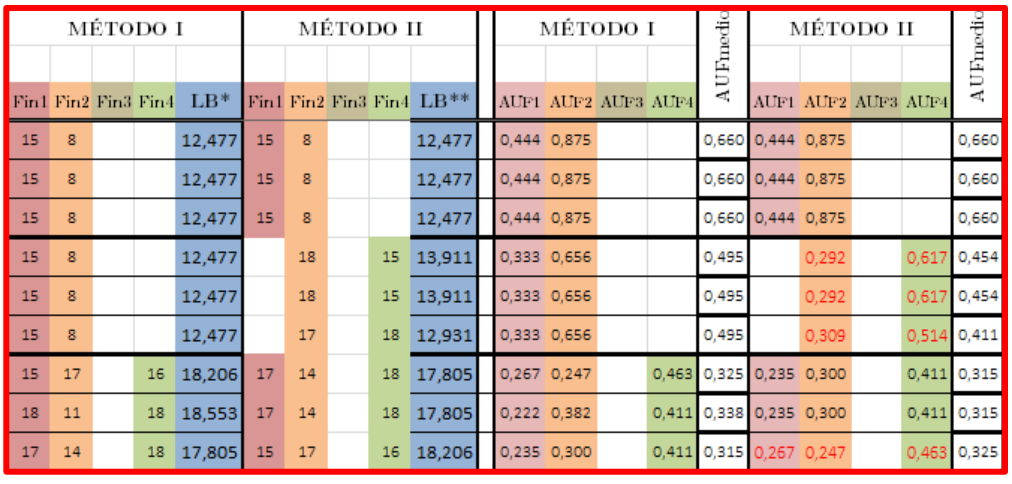
Figura. Ejemplo de las anotaciones realizadas para una cartera

instancia				Hor	Rec	s	AUF** iniciales			
271	1	2	4	18	3	0.01	0,370	0,389		0,685
272	1	2	4	18	3	0.02	0,370	0,389		0,685
273	1	2	4	18	3	Var	0,370	0,389		0,685
274	1	2	4	18	4	0.01	0,278	0,292		0,514
275	1	2	4	18	4	0.02	0,278	0,292		0,514
276	1	2	4	18	4	Var	0,278	0,292		0,514
277	1	2	4	18	5	0.01	0,222	0,233		0,411
278	1	2	4	18	5	0.02	0,222	0,233		0,411
279	1	2	4	18	5	Var	0,222	0,233		0,411

Detalle I

Proyectos que componen la cartera inicial ($\{1, 2, 4\}$), horizonte temporal (Hor: 18), recurso global (Rec: 3, 4, 5 unidades) y pasos (s: 0.01, 0.02, variable).

AUF iniciales de los proyectos, si concluyen en su tiempo de camino crítico.



MÉTODO I					MÉTODO II					MÉTODO I					MÉTODO II				
Fin1	Fin2	Fin3	Fin4	LB*	Fin1	Fin2	Fin3	Fin4	LB**	AUF1	AUF2	AUF3	AUF4	AUFmedio	AUF1	AUF2	AUF3	AUF4	AUFmedio
15	8			12,477	15	8			12,477	0,444	0,875			0,660	0,444	0,875			0,660
15	8			12,477	15	8			12,477	0,444	0,875			0,660	0,444	0,875			0,660
15	8			12,477	15	8			12,477	0,444	0,875			0,660	0,444	0,875			0,660
15	8			12,477		18		15	13,911	0,333	0,656			0,495		0,292		0,617	0,454
15	8			12,477		18		15	13,911	0,333	0,656			0,495		0,292		0,617	0,454
15	8			12,477		17		18	12,931	0,333	0,656			0,495		0,309		0,514	0,411
15	17		16	18,206	17	14		18	17,805	0,267	0,247		0,463	0,325	0,235	0,300		0,411	0,315
18	11		18	18,553	17	14		18	17,805	0,222	0,382		0,411	0,338	0,235	0,300		0,411	0,315
17	14		18	17,805	15	17		16	18,206	0,235	0,300		0,411	0,315	0,267	0,247		0,463	0,325

Detalle II

Según los dos métodos, fechas de finalización de la cartera (o subcartera) solución en la programación. Valor de la cartera (LB).

Cálculo en las dos opciones de los valores de AUF para cada proyecto de la cartera solución y el AUF_{medio} de todos ellos.

				Complejidad de la red			
NPVmax /RU				NC1	NC2	NC3	NC4
0,300	0,381		0,243	1,05	1,29		1,61
0,300	0,381		0,243	1,05	1,29		1,61
0,300	0,381		0,243	1,05	1,29		1,61
0,300	0,381		0,243	1,05	1,29		1,61
0,300	0,381		0,243	1,05	1,29		1,61
0,300	0,381		0,243	1,05	1,29		1,61
0,300	0,381		0,243	1,05	1,29		1,61
0,300	0,381		0,243	1,05	1,29		1,61
0,300	0,381		0,243	1,05	1,29		1,61

Detalle III

Relación $NPV_{máx}/RU$ para los proyectos iniciales en la cartera y valor de la complejidad de la red para cada uno.

Como ya dijimos, tenemos un total de once carteras diferentes, cada una de ellas ejecutada para cuatro horizontes temporales distintos (el máximo tiempo de camino crítico de los proyectos de la cartera inicial, y los otros tres niveles aumentando en un 10%, en un 20% y en un 30%), para tres valores límite de recurso global (3, 4, y 5 unidades) y para tres pasos diferentes de actualización de precios (0.01, 0.02 y el paso variable que introdujimos en la programación). Teniendo en cuenta que tenemos dos programaciones a comparar (método I y método II), calculamos el total de ejecuciones que hicimos:

$$11\ carteras\ \times 4\ horizontes \times 3\ límites\ de\ recurso \times 3\ pasos\ \times 2\ métodos = 792\ ejecuciones$$

Aunque nuestro análisis en concreto no se pueda tratar como un análisis de experimentos propiamente (debido a las características de los datos, aunque sí que pudiéramos ver que cada instancia se repite en varios niveles, la función respuesta no es la que habitualmente podemos tener, puesto que no es una variable SI/NO que nos pudiera hacer pensar en una regresión logística, o no se comporta como una distribución normal para poder hacer un diseño de experimentos como lo conocemos. La particularidad de nuestra respuesta (la cartera solución y su valor) nos hace desestimar este tipo de análisis más habituales), cabe resaltar que no tendría sentido ejecutar varias veces la misma cartera con los mismos parámetros. La razón es que, como se ha podido apreciar al describir el funcionamiento del algoritmo en los capítulos IV y V (y si no es así, se le aclara en este momento al lector), toda la programación es de carácter determinista, esto es, siempre que ejecutemos la misma instancia, obtendremos la misma solución, ya que nada depende del azar, probabilidades o decisiones del que ejecuta la aplicación. Incluso en el caso de que quisiéramos medir número de iteraciones o tiempo de ejecución, los resultados obtenidos serían los mismos. Por ello, repetimos como en nuestro análisis no es necesario un número de réplicas de cada instancia.

VI.b. Conclusiones obtenidas

Como ya anticipábamos, la particularidad de los datos que podemos obtener al ejecutar la aplicación, no son los que habitualmente podrían estar presentes en un análisis estadístico común.

La variable respuesta se trata del valor que a la organización reportaría esa programación descrita por los tiempos de finalización de cada proyecto que se ejecuta en la subcartera solución, lo que hace que no pueda seguir ningún patrón regular, o una relación entre las soluciones de cada instancia (de hecho, depende directamente de los valores $NPV_{máx}$ que demos a cada proyecto). En su momento, para intentar solventar esta problemática, nos planteamos el traducir esta respuesta a una variable booleana, de tal forma que valiera 1 si se había encontrado la solución óptima, o 0 en el caso contrario, pero nos encontrábamos de nuevo con un problema: ¿cómo asegurar que se trata de la programación óptima?

Incluso con proyectos pequeños como los que tenemos, no podemos confirmar en todas las combinaciones de carteras que tenemos, que la aplicación esté encontrando la mejor solución. De hecho, si ya desde un primer momento supiéramos cuál es la solución óptima, no recurriríamos a ninguna aplicación y no estaríamos abordando el tema de este informe.

Por ello, sabemos que nuestro programa nos va a dar soluciones factibles, ejecutables por la organización y que, en la buena mayoría de los casos, corresponderán con las soluciones óptimas, si no buenas aproximaciones, pero no podemos recurrir a esa variable booleana porque no podremos asegurarlo.

Por lo tanto, viendo en el apartado anterior con qué parámetros hemos contado y las anotaciones que tenemos de cada instancia, terminaremos contestando a preguntas del tipo:

- ¿Hay alguna razón aparente para que un proyecto entre a la subcartera solución y otro no?
- ¿Se obtienen mejores resultados con alguno de los tres pasos para la actualización de precios?
- ¿Alguno de los dos métodos (caracterizados por su regla de prioridad en el método serie) da mejores resultados sistemáticamente?
- Cuando el método II nos programa otra solución, ¿tiene mejor aprovechamiento de recursos (AUF_{medio})?

Intentaremos en las próximas páginas ir valorando estas preguntas y justificar nuestras conclusiones.

Selección de proyectos

Para intentar abordar la cuestión de por qué algunos proyectos son excluídos de la subcartera solución, de las 396 instancias que hemos ejecutado, nos quedaremos con los resultados de aquellas

en las que algunos de los proyectos iniciales sí que han sido desestimados. Analizaremos ambos métodos por separado teniendo en cuenta algunos de los parámetros ya mencionados en el apartado anterior.

En primer lugar vamos a valorar lo que hemos llamado *AUFiniciales* (recordamos que propiamente el término *AUF* se vincula a cuando el proyecto ya está dentro de la programación, pero el concepto es similar).

¿Si un proyecto en su programación de camino crítico utiliza mayor cantidad del total de recursos que ponemos a su disposición, tiene más probabilidades de entrar en la cartera solución?

Presentamos los resultados gráficamente. En el método I, de los 396 problemas, en un total de 315 no se terminan programando todos los proyectos de la cartera inicial, mientras que en el método II en 298. De ese total, hacemos dos clasificaciones: las soluciones que sí que toman las subcarteras con mayores *AUFiniciales*, y las que no lo hacen.

Esto supone:

- **Método I:** el 60,6 % de los problemas toman en su solución final los proyectos con mayor valor de *AUFinicial*.
- **Método II:** supone el 78,2 %.

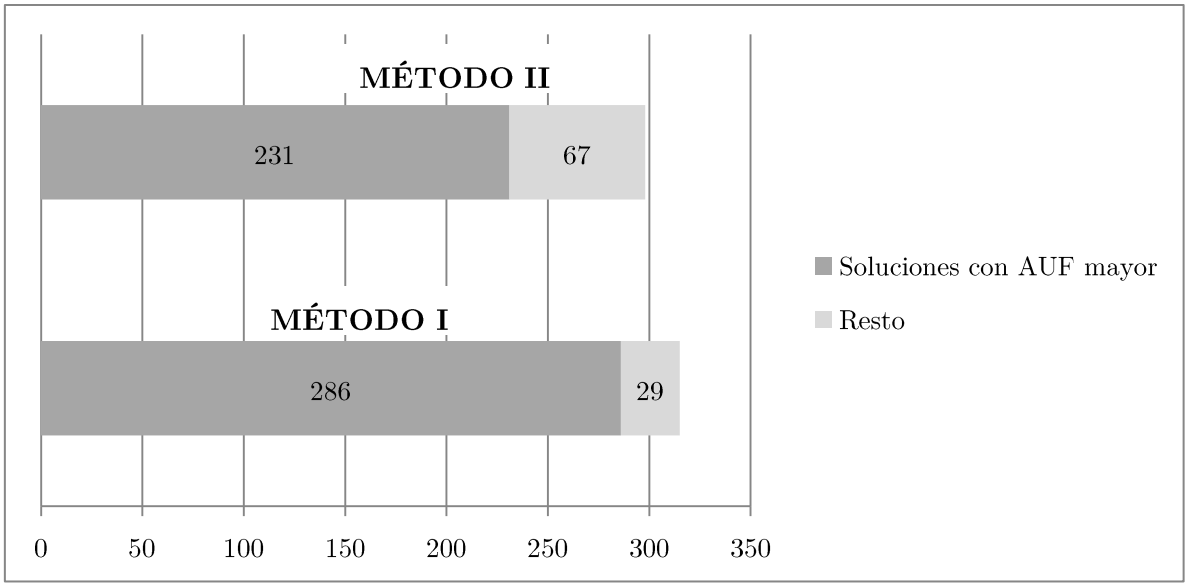


Figura. Comportamiento en la selección respecto al *AUFinicial*

El hecho de que ya de por sí el método II tenga menor número de instancias en donde no se terminan programando todos los proyectos de la cartera inicial, ya nos habla de una mayor efectividad, puesto que consigue más programaciones incluyendo a todos los proyectos.

Por otra parte, al tener un 78,2 % frente a un 60,6 % del método I, sus soluciones incluyen en más ocasiones a proyectos que optimizan más el uso de recursos dentro de su camino crítico.

Cabe destacar que el hecho de que un proyecto sea incluído o no, puede verse condicionado por su valor de $NPV_{máx}$, puesto que al fin y al cabo, la solución final se rige por el mayor LB, por ello, si valoramos una cartera de dos proyectos, en la que por limitaciones de horizonte temporal sólo puede ejecutarse uno de ellos, es evidente que no "vencerá" quién tenga un mejor AUF inicialmente, si no que lo hará quién tenga un mayor valor como proyecto (si en una iteración se consigue programar el que tiene mayor $NPV_{máx}$, ésta va a ser la solución que nos dé la aplicación).

El otro factor que puede influenciar en la selección de proyectos para la subcartera solución es el cociente $NPV_{máx}/RU$. Como ya dijimos es una especie de rentabilidad del proyecto: cuánto es lo máximo que me podría reportar frente al número de unidades que debo consumir para su ejecución.

Haremos un tratamiento similar al anterior desde el gráfico:

- **Método I:** el 90,8 % de las soluciones incluye los proyectos que tienen mayor ratio $NPV_{máx}/RU$.
- **Método II:** supone el 77,5 %.

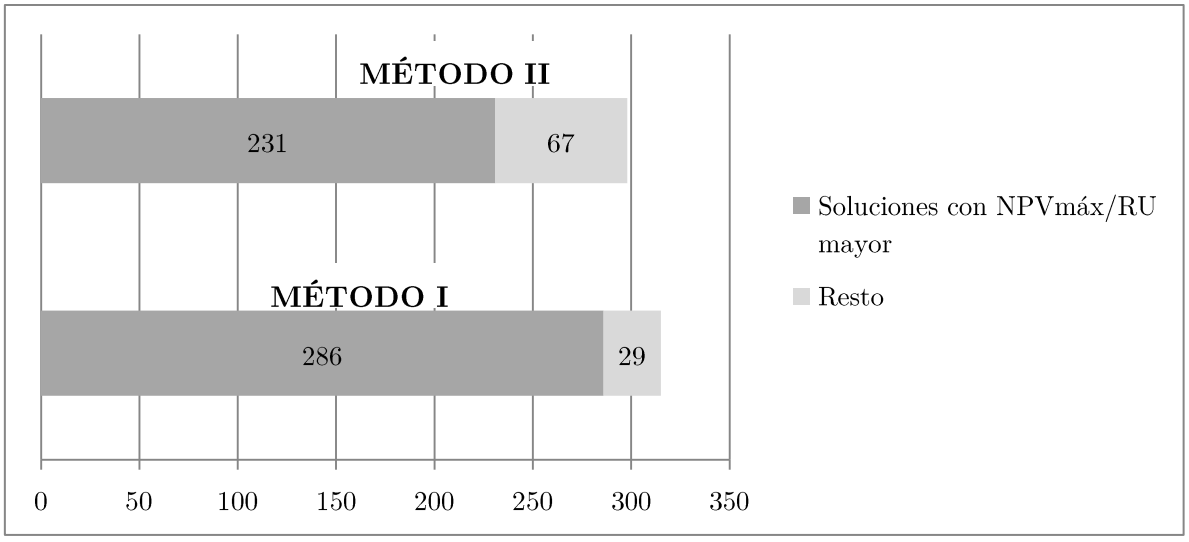
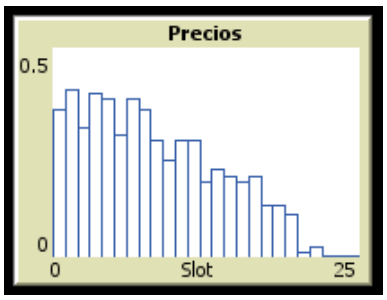


Figura. Comportamiento en la selección respecto al $NPV_{máx}/RU$

La apreciación que sacamos en este caso es que, el método I, en detrimento de obtener el valor global de la cartera más alto, incluye en más ocasiones los proyectos que hemos descrito como más rentables. El método II, por el contrario, consigue soluciones con valores globales superiores de la cartera final (le condiciona más solamente el valor de $NPV_{máx}$).

Elección del paso para actualización de precios



Como ya vimos, la convergencia del algoritmo depende directamente del equilibrio de precios en cada uno de los slots del horizonte temporal que marquemos. El llegar a esta situación está directamente vinculado al paso que elijamos. Influye en la rapidez de la obtención de la solución, e incluso, al tener un criterio de parada en relación al número de iteraciones, el terminar encontrando la solución o no.

Recordamos que en nuestro caso teníamos dos pasos fijos (0,01 y 0,02) los cuáles nos venían como referencia en el artículo, y planteábamos un paso variable en el que, dependiendo de si había más demanda u oferta de recursos en un slot, el precio subiría o bajaría, respectivamente.

La pregunta clave en relación a los pasos es evidente: ¿podemos confirmar que funciona mejor el paso fijo o el variable en este algoritmo?

Para ello, observaremos los valores finales de las carteras solución y veremos con qué paso se obtienen valores más altos. En este caso, pretendemos hacer conclusiones comunes a ambos métodos, es decir, saber que de forma general una opción de paso es mejor que otra, independientemente de la heurística de prioridad en el método serie de la programación.

Los resultados son los siguientes:

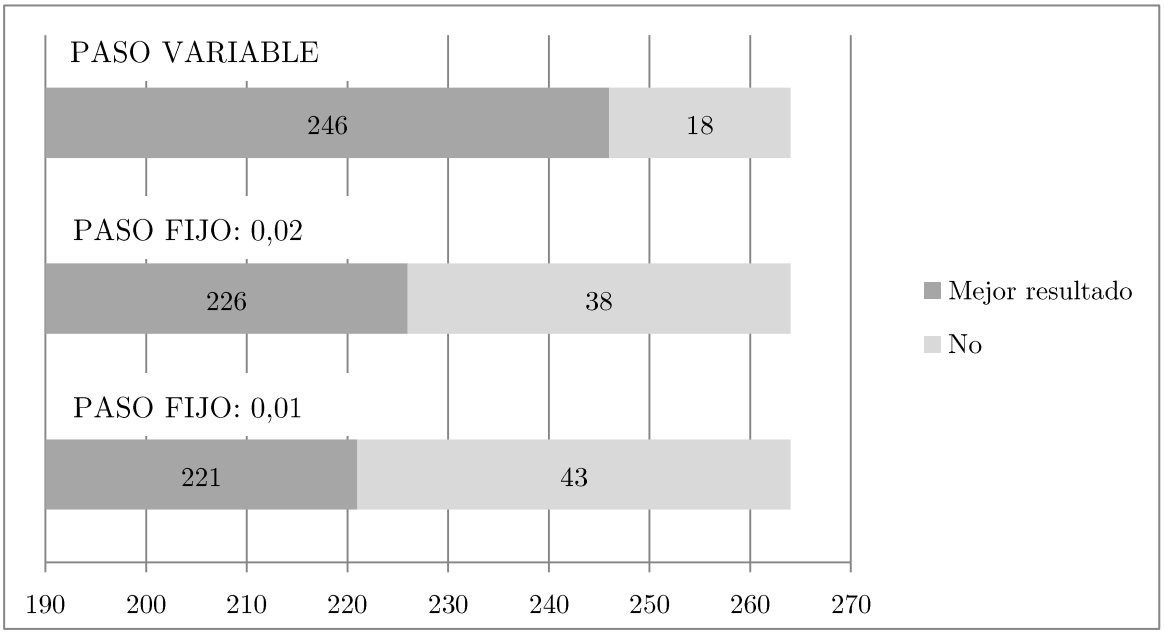


Figura. Comportamiento del algoritmo frente al paso (s)

Como ya dijimos, pretendemos sacar una conclusión general, y no asociada directamente a uno de los dos métodos, por ello de las 792 ejecuciones hechas, se tienen 264 con cada paso (la tercera parte). Por ello, en el gráfico mostramos de esas 264 veces, cuántas se han conseguido la solución de mayor beneficio.

Aunque gráficamente ya podemos concluir con qué paso nos ha dado una mejor respuesta, añadimos como dato el porcentaje de éxitos:

Paso variable	**93,2 %**
Paso fijo (s = 0,02)	85,6 %
Paso fijo (s = 0,01)	83,7 %

Por tanto, concluímos con que el funcionamiento del algoritmo se ve favorecido con el uso del paso variable. Esta opción sí que ha sido utilizada en varias ocasiones por multitud de autores, y como vemos, en nuestro experimento consigue las carteras de mayor valor con un alto porcentaje.

Luego, podemos confirmar que uno de los primeros cambios que incorporamos en su momento al algoritmo, ha terminado siendo una mejora significativa.

Comparativa de resultados entre método I y método II

Quizás esta parte del análisis puede ser la más relevante, puesto que nos permitiría confirmar que las variantes incorporadas al algoritmo original del artículo, sí que consiguen mejores resultados.

Recordamos que las diferencias que incorporábamos dentro de la réplica del algoritmo, están detalladas en el apartado V.d. del presente informe. Principalmente, lo que iba a condicionar los resultados era la regla del criterio de prioridad en el método serie: mientras que en el artículo se tomaba un criterio basado en tiempos de las actividades (su fecha de inicio si se programara sin restricciones y su duración), el nuevo criterio que proponemos incluía el valor del proyecto. Es decir, en principio conseguimos que una actividad sea más relevante si pertenece a un proyecto cuya finalización nos dará más valor a la subcartera final.

En primer término, la comparativa que vamos a hacer es sencilla: partiendo de que ambos métodos reportan soluciones factibles, que podrían ser acometidas perfectamente por la organización, ¿qué método encuentra soluciones finales de mayor valor?

Vamos a resumir cuáles son las variaciones en cuento a resultados de ambos métodos. Podemos decir que, de entre las 396 carteras ejecutadas por ambos métodos:

- En 94 ocasiones los resultados encontrados por el método I son distintos que los del método II (en este caso, aún no particularizamos si son mejores o peores soluciones). Esto constata

cómo la modificación del método serie, efectivamente, condiciona las programaciones finales. En nuestro caso, en casi un 25 % de las instancias.

- El método II consigue en 386 ocasiones, los mismos resultados o mejores que el método I. Por tanto, sólo en 10 problemas ejecutados, el método II encuentra alguna programación con menor valor final. Cierto es que todos estos datos podrían verse modificados si el criterio de parada decidido en la programación fuera menos restrictivo, puesto que con mayor número de iteraciones se podría alcanzar la combinación de precios de la solución óptima en ambos casos.

- El método II en 84 ocasiones encuentra mejores soluciones, lo que supone un 21,2 % de mejora gracias al nuevo método.

												MÉTODO I					MÉTODO II				
instancia					Hor	Rec	s	AUF** iniciales				Fin1	Fin2	Fin3	Fin4	LB*	Fin1	Fin2	Fin3	Fin4	LB**
394	1	2	3	4	20	5	0.01	0,200	0,210	0,270	0,370	16	8	14		22,241		19	12	19	23,081
395	1	2	3	4	20	5	0.02	0,200	0,210	0,270	0,370	16	9	12		22,883		19	12	19	23,081
396	1	2	3	4	20	5	Var	0,200	0,210	0,270	0,370	16	9	12		22,883		19	12	19	23,081

Figura. Ejemplo de mejor programación encontrada por el método II respecto al I

Analizamos el caso particular representado en la imagen. Se trata de tres carteras formadas por los cuatro proyectos, permitiendo un horizonte temporal de 20 unidades y un límite de recurso global en cada slot de 5 unidades. Como vemos en la columna azul, para los tres pasos de actualización de precios, el método II encuentra soluciones finales de mayor valor (23,081 unidades). Mientras que el método I nos proponía ejecutar los proyectos $\{2, 3, 1\}$ con los tiempos de finalización correspondientes, el método II propone la subcartera de proyectos $\{3, 2, 4\}$.

Por completar, vamos a ver para este ejemplo las salidas de la aplicación Netlogo ($s = 0{,}01$) con las programaciones propuestas que recogíamos en la tabla:

MÉTODO I

MÉTODO II

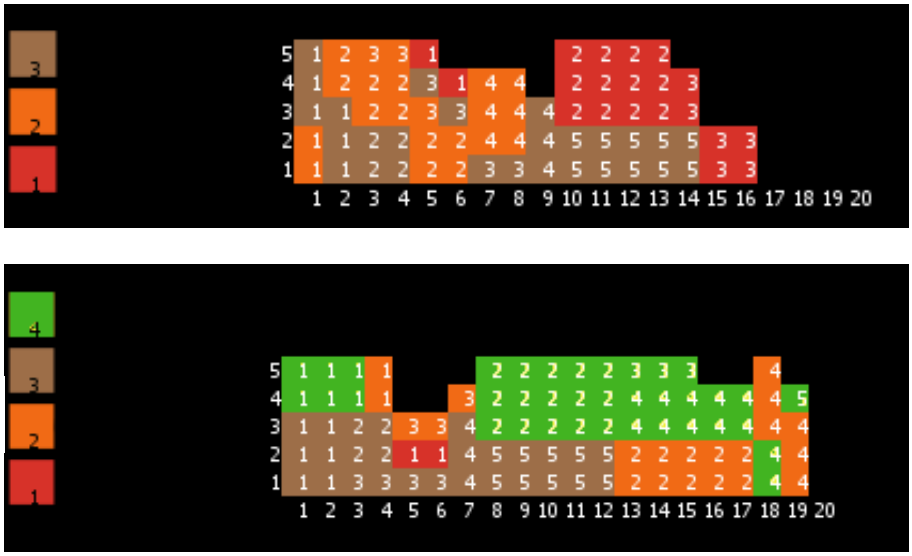


Figura. Ejemplo de dos programaciones diferentes propuestas por método I y método II para una misma cartera de origen

Luego, teniendo las conclusiones de este apartado, podemos determinar cómo el método II se trata de una mejor elección, puesto que en un 97,5% de las carteras ha encontrado las mismas soluciones que el método original o mejores, luego la probabilidad de obtener la solución óptima o próxima a ella, se vería aumentada utilizándolo.

Comportamiento del AUF_{medio} para ambos métodos

Este es el último gran punto en el que nos hemos centrado para valorar los dos métodos: cuando los dos métodos no nos dan la misma solución, ¿se ve que alguno da soluciones con mayor AUF_{medio}? Es decir, saber que las subcarteras solución están saturando más o menos los recursos que la organización pone a su disposición.

Como ya vimos en la descripción del parámetro, el concepto del valor medio del AUF para los proyectos solución, pretende ser una medida de efectividad ya que, de alguna forma, estamos diciendo que los recursos dispuestos por la organización son mejor aprovechados. Bien es cierto, que perseguir un buen AUF_{medio} no debe ir en detrimento de encontrar una solución lo más beneficiosa posible, porque ninguna de nuestras pautas iniciales nos dice que debamos prioridad una solución más eficiente frente a una que reporte más valor.

En el anterior apartado ya comentamos como hay un total de 94 casos en los que el método I y el II nos proporcionan soluciones diferentes. Solamente en esas carteras es donde tiene sentido plantearnos la variabilidad del parámetro.

Aprovechando el ejemplo que propusimos en el anterior apartado, y ya que ambos métodos daban programaciones diferentes, vamos a calcular para cada uno su AUF_{medio}. Como vimos en la tabla, la limitación de recurso global en este ejemplo era de 5 unidades (necesario para el cálculo de recursos disponibles).

SOLUCIÓN **MÉTODO I**					
Subcartera	$t_{finalización}$	RU	$R_{disponibles}$	AUF_i	AUF_{medio}
Proyecto 2	8	21	$5 \times 8 = 40\ unidades$	$\frac{21}{40} = 0{,}525$	$\frac{0{,}525 + 0{,}386 + 0{,}25}{3} = 0{,}387$
Proyecto 3	14	27	$5 \times 14 = 70\ unidades$	$\frac{27}{70} = 0{,}386$	
Proyecto 1	16	20	$5 \times 16 = 80\ unidades$	$\frac{20}{80} = 0{,}25$	

SOLUCIÓN **MÉTODO II**					
Subcartera	$t_{finalización}$	RU	$R_{disponibles}$	AUF_i	AUF_{medio}
Proyecto 3	12	27	$5 \times 12 = 60\ unidades$	$\frac{27}{60} = 0{,}45$	$\frac{0{,}45 + 0{,}221 + 0{,}389}{3} = 0{,}353$
Proyecto 2	19	21	$5 \times 19 = 95\ unidades$	$\frac{21}{95} = 0{,}221$	
Proyecto 4	19	37	$5 \times 19 = 95\ unidades$	$\frac{37}{95} = 0{,}389$	

Luego, en esta ocasión vemos como la programación propuesta por el método II tiene un valor del parámetro AUF_{medio} menor que en el método I, pero en detrimento de un mayor valor de la solución final, como ya vimos.

Si nos fijamos en este ejemplo concreto, vemos como tiene sentido que esta media sea menor, puesto que el método II propone (dentro del horizonte temporal límite que eran 20 unidades) una programación mucho más dispersa a lo largo del horizonte, teniendo a dos proyectos que finalizan en el slot 19. Esto hace que la saturación de uso de recursos en cada slot, sea irremediablemente menor.

Pues bien, hicimos un cálculo equivalente para esas 94 carteras en las que se obtenían soluciones diferentes y lo obtenido fue lo siguiente:

- Exactamente el 50 % de los casos (47 ocasiones), el método I reporta soluciones con el AUF_{medio} mayor, por lo que la otra mitad de las carteras tienen mayor valor del parámetro al ejecutar el método II.
- De las 47 instancias en las que el método II tiene un AUF_{medio} menor, en 41 de ellas la solución que propone tiene mayor valor que la del método I.

Es decir, en el 87,2 % de las ocasiones en las que el método II propone programaciones menos saturadas que el método I, lo hace en detrimento de obtener un mayor valor para la cartera.

Por tanto, lo que podemos concluir en este caso es que el valor medio de utilización de recursos en la subcartera solución no parece determinante y no se puede afirmar que ninguno de los dos métodos esté favoreciendo a este parámetro.

Además, el método II persigue soluciones de mayor valor, que ha sido siempre el objetivo que se ha perseguido, luego, quizás no haya un mayor condicionante para las programaciones que obtiene que el valor máximo de los proyectos, puesto que será lo que condicione su inclusión en la subcartera solución.

Tras analizar estos apartados de forma individual, podemos resumir con que las incorporaciones introducidas en el algoritmo original que se defendía en el artículo, han resultado ser positivas para el funcionamiento del mismo: la nueva regla de prioridad consigue soluciones de mayor valor para la organización si no las mismas que obtenía el método original, con una mejor convergencia gracias al paso variable en la actualización de precios y, aunque nunca fuera condicionante, las salidas propuestas por el método II no parecen suponer un desperdicio de recursos para la organización.

Además, podemos asegurar que el valor final que reporta la subcartera solución es más correcto y a la vez realista, que el método de valoración que utilizaban en el artículo, lo cuál también es importante ya que, permite que las previsiones que podamos hacer sean los más acordes posibles con el futuro. Si en nuestras simulaciones utilizamos un k=0,05, la aplicación está preparada para la modificación en el panel de inicio de este valor, lo que también permite una variable más susceptible de tener en cuenta en futuros análisis.

VII. ESTUDIO ECONÓMICO DEL PROYECTO

Sea cual sea el proyecto que se realice, siempre es básico que esté acompañado de un análisis económico. Por ello, en este apartado detallaremos qué personal ha estado involucrado en la realización de este proyecto, los costes asociados y la posible rentabilidad económica y salida que podría tener la aplicación diseñada.

Cabe destacar que, en cualquier caso, realizamos lo que sería una estimación y todo ello suponiendo posibles ventas a lo largo de cuatro años desde que este proyecto se vea finalizado.

Por otra parte, se tendrá en cuenta un concepto muy presente sobre todo en la industria tecnológica y de la investigación y el desarrollo, con es el "know-how". La experiencia, relaciones establecidas en el mercado o el conocimiento por expertos de tu organización en lo que comercializas, aún siendo bienes intangibles, también están a disposición de los clientes que consumen tu producto, y por ello, tienen que tener también su representación en los costes que imputes a tu trabajo. En este capítulo pondremos precio a esta aportación para que el proyecto sea rentable en el horizonte temporal que valoramos.

VII.a. Personal involucrado

El desarrollo de este PFC que aporta una aplicación en Netlogo para la programación y selección de proyectos en una cartera, ha tenido, además de al ingeniero industrial pertinente, la colaboración de un director y un co-director. Las misiones que ha desempeñado cada uno podrían quedar resumidas como:

- Director: Dr. Adolfo López Paredes. Además de proponer la idea del mismo, ha sido encargado de gestionar los hitos más importantes del proyecto, valorar las posibles alternativas en las que dirigir el proyecto y supervisar el buen progreso y su correcta documentación en este informe.

- Co-director: Félix Antonio Villafáñez Cardeñoso. En su labor complementaria, ha atendido aquellos problemas más técnicos o conceptuales que se pudieran plantear a lo largo del desarrollo de la aplicación. Su experiencia en el campo, y trabajando en Netlogo, le ha permitido aportar consejos previos que han evitado posibles complicaciones posteriores.

- Ingeniero Industrial: Carolina Rodríguez Pulgar. Encargada del desarrollo de la aplicación y el informe en cuestión, asesorada por director y co-director.

VII.b. Estudio de costes

Los costes atribuibles al desarrollo del PFC en cuestión, se pueden descomponer en el salario bruto, la cantidad asociada a beneficios sociales y el uso de infraestructuras. En todos los casos, para facilidad en el cálculo, el coste lo consideraremos en función de las horas de trabajo. Tendremos en cuenta las horas de desarrollo de todo el proyecto, así como las sesiones de reunión para valorar la evolución.

En primer lugar, desglosamos los costes vinculados al director. Como catedrático en el departamento de organización y C.I.M de la Universidad de Valladolid, estimamos que el precio de su hora de trabajo está valorado en torno a 110 €/hora. Por otra parte, las reuniones citadas con el ingeniero en cuestión, han sido un total de 3, con duración aproximada de una hora. Las reuniones de valoración con el co-director han sido otras 3 con la misma duración. Por último, la revisión y corrección del informe se estima en otras 3 horas.

Consideramos que el coste por beneficios sociales, termina siendo prácticamente la misma cantidad que el salario bruto, y que el uso de infraestructuras (luz, equipos informáticos) supone la mitad del coste salarial. Este planteamiento lo usaremos exactamente igual en el caso de los otros dos trabajadores. Así:

DIRECTOR	€ / hora	Horas	TOTAL
Salario Bruto	110		990
Beneficios sociales	110	9	990
Infraestructuras	55		495
			2.475 €

Por otra parte, el co-director ha realizado las reuniones con el director y un total de 4 sesiones de revisión del desarrollo de la programación de en torno a una hora. Atribuímos, en este caso, un salario bruto de 90 €/hora. Por lo que su trabajo queda valorado en:

CO-DIRECTOR	€ / hora	Horas	TOTAL
Salario Bruto	90		630
Beneficios sociales	90	7	630
Infraestructuras	45		315
			1.575 €

En el caso del ingeniero, podremos estimar las horas de trabajo valorando las diferentes etapas para la realización del PFC. Su sueldo se asume en 50 €/hora.

	Días	Horas / día	TOTAL
Documentación	45	4	180
Programación y análisis	60	5	300
Realización del informe	30	4	120
			600 h

INGENIERO	€ / hora	Horas	TOTAL
Salario Bruto	50		30.000
Beneficios sociales	50	600	30.000
Infraestructuras	25		15.000
			75.000 €

Luego, sumando el trabajo del personal, la ejecución del PFC se estima en un total de:

$$2.475 + 1.575 + 75.000 = \mathbf{79.050}\,€$$

VII.c. Rentabilidad económica del proyecto y comercialización

Una vez valorado los costes para el desarrollo de la aplicación, planteando las posibles ventas en un horizonte temporal de 4 años, vamos a estudiar la rentabilidad que tendría este proyecto.

Si bien es cierto, hasta el momento sólo hemos valorado las horas de trabajo que ha supuesto para el equipo llevar a cabo este proyecto, pero también debemos tener en cuenta ese conocimiento intangible que nuestra experiencia o el saber exactamente los entresijos de la programación está dentro de nuestro equipo y que, al fin y al cabo, también quedan a disposición del cliente cuando reclama nuestro servicios. Es el concepto del "know-how". ¿Cómo podemos estimar el valor de este conocimiento que también "vendemos"? En este apartado veremos en cuánto tendríamos que valorarlo, según las ventas estimadas, para que este proyecto sea rentable en el horizonte temporal propuesto.

Vamos a diferenciar las posibles ventas entre PYMES y grandes empresas, ya que las horas de trabajo que la adaptación de la aplicación nos supondría, serán diferentes según el cliente en cuestión (es probable que una gran empresa tenga muchos más posibles proyectos a acometer y que por ello, el peso computacional y las variables de cálculo sean mucho más importantes).

Desde la visión más realista posible, este primer año, ya avanzado, no se considera que podramos llegar a grandes empresas, e incluso se trataría de una mejor opción comercial, ya que podríamos testar mejor los primeros resultados al comenzar a comercializar el producto. De alguna forma, la ventaja que tendrán los clientes que vengan en los próximos años es contar cada vez con mayor experiencia por nuestra parte y tendrán esa ventaja diferencial.

VENTAS (ud)	2013	2014	2015	2016
PYMES	2	3	4	4
Grandes empresas	-	1	1	2

Horas	Días	Horas/día	TOTAL
PYMES	25	8	**200 h**
Grandes empresas	60	8	**480 h**

Vemos en la segunda tabla, las horas de trabajo que la modificación para cliente nos podrá suponer. Por ello, el total de horas dedicadas cada año en el proyecto temporal, son:

Horas	2013	2014	2015	2016
PYMES	400	600	800	800
Grandes empresas	-	480	480	960

Para poder incorporar en el modelo el concepto del "know-how" podemos multilicar las horas en cuestión por un valor X (en unidades €/hora) que será ese valor añadido que nuestro conocimiento y experiencia da a cada proyecto. Si consideramos una tasa impositiva t del 20%, los flujos de caja para el cálculo de la rentabilidad serán:

$$FCF = horas_{año} \times X \times (1 - t)$$

Luego, los flujos de caja para cada año serán los siguientes:

FCF	2013	2014	2015	2016
PYMES	320X	480X	640X	640X
Grandes empresas	-	384X	384X	768X
	320X	864X	1024X	1408X

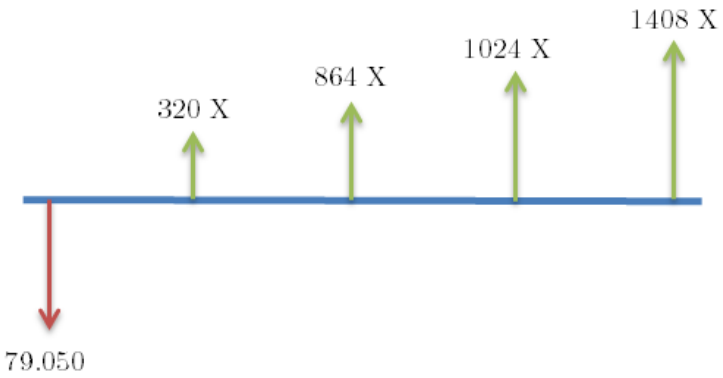


Figura. Representación de inversión y FCF del proyecto

Si suponemos una tasa de descuento k constante todos los años de en torno al 10% (la rentabilidad de un bono en el mercado actual, con la misma duración que el proyecto y preferiblemente en nuestra misma moneda), a través de la definición del valor actual neto (VAN), podemos calcular el valor límite de X para que el proyecto sea rentable:

$$VAN = -A + \sum \frac{FCF_i}{(1+k)^i}$$

$$-79.050 + \frac{320X}{(1+0.1)} + \frac{864X}{(1+0.1)^2} + \frac{1024X}{(1+0.1)^3} + \frac{1408X}{(1+0.1)^4} = 0$$

$$-115737.105 + 425.92X + 1045.44X + 1126.4X + 1408X = 0$$

$$\boxed{X = 28.89\ €/hora}$$

Por tanto, si se cumplen las expectativas de ventas, con una representación de aproximadamente 30 €/hora para el "know-how", podríamos asegurar la rentabilidad de la comercialización de la aplicación. Si el precio por la hora de trabajo del ingeniero se sigue manteniendo en ese total de 125 €/hora, tendríamos que calcular presupuestos para los clientes en torno a los 155 €/hora.

VIII. CONCLUSIONES Y FUTURAS LÍNEAS DE INVESTIGACIÓN

VIII.a. Resumen del informe

En este último capítulo pretendemos resumir los puntos más relevantes del PFC que se recoge en este documento, además de remarcar algunas de las ideas y conocimientos más importantes que el autor de este informe ha podido extraer de la realización de este proyecto.

El objetivo de la realización del proyecto fin de carrera en una titulación pretende demostrar cómo el alumno puede llegar a utilizar algunos de los conocimientos aprendidos en estos años e incluso ampliarlos y ponerlos en práctica en una aplicación directa del campo de la ingeniería. En nuestro caso en particular, dentro del mundo de la gestión y organización de empresas, encontramos como la temática que aborda este proyecto es de total actualidad y utilidad en las organizaciones.

El Project Scheduling es una rama en desarrollo que tiene como objetivo principal optimizar recursos, tiempos y por tanto, beneficios en una organización. Es habitual que en cada vez un mayor número de empresas los trabajos se vayan acometiendo en forma de proyectos, de tal forma que disponemos de unos recursos, para realizar unas actividades (que en ocasiones, tendrán una planificación determinada, a la cuál tendremos que prestar especial atención), un tiempo límite para verlos realizados e incluso, una financiación a la que vernos sometidos.

Si ya disponemos de varios proyectos simultáneos (lo que denominamos, cartera de proyectos) deberemos conjugar todos ellos, teniendo en cuenta todas las variables que llegan a estar involucradas, para siempre obtener de la forma más rápida, la solución más óptima y por supuesto, factible. Esta solución puede ser la programación en el tiempo de todos ellos, o bien una selección de aquellos que son más interesantes según nuestro criterio. Ahí entrará la función, cada vez más valorada, del Project Manager, el cuál deberá con múltiples herramientas a su disposición, determinar cuáles son los intereses de la organización y qué soluciones abordar.

Dentro de este área de conocimiento, este proyecto fin de carrera prentende dar un paso más en la investigación de las múltiples metodologías para programación y selección de carteras de proyectos. Por ello, en los primeros temas nos encargamos de resaltar las vertientes más importantes que se están barajando dentro de la programación de proyectos y entender la cantidad de variables que se deben tener en cuenta.

Una publicación en 2010 en la revista Springer, titulada "*Combinatorial auction algorithm for project portfolio selection and scheduling to maximize the net present value" (Shou & Huang)*, suscita el siguiente informe. En ella se propone un algoritmo para optimización y selección de carteras de proyectos, del cuál realizamos una réplica en el programa de simulación de agentes

Netlogo, y analizamos cuáles son sus puntos claves, pero también sus puntos susceptibles de mejorar o debilidades.

Lo que caracteriza a esta propuesta es sobre todo, el uso de las subastas combinatorias. Ellos conciven a los proyectos como pujadores, de tal forma que un ente superior termina dirigiendo una subasta donde, dependiendo de la demanda y oferta de los recursos en cada instante del horizonte (slot), los precios subirán o bajarán. En el caso de que en un slot la demanda sea superior a la disponible, aumentará su precio, mientras que si sobra recurso disponible, bajarán los precios en ese slot. Con esto conseguimos que los proyectos pujen por una combinación de varios recursos (por ello el término de combinatoria) y que sus actividades se vayan desplazando en el horizonte temporal según su conveniencia, hasta llegar, tras varias iteraciones, a un equilibrio de precios que nos dará la solución óptima para esa cartera.

De la aplicación desarrollada, gracias a Netlogo, se permite ver una representación muy visual del funcionamiento interno del algoritmo, lo que resulta ser incluso ilustrativo para aquél que ejecute el programa.

Por otra parte, el entendimiento pormenorizado de todos los pasos del algoritmo, nos permite tener una visión más crítica respecto a su propuesta y saber qué podemos aportar que mejore sus resultados.

Dentro de su proposición inicial (resumen en diagrama de bloques en el apartado IV.g.), terminamos modificando dos de los puntos más relevantes en el programa: la valoración de proyectos y la heurística de asignación de prioridades que ordena las actividades de la solución final.

Con respecto a la valoración de los proyectos, encontrábamos como en el artículo simplemente se reducía el valor máximo del proyecto en un porcentaje fijo dependiendo de lo que se alejara de su tiempo mínimo de ejecución. Si el valor final del proyecto lo consideramos como un flujo de caja más para la organización, bien sabemos que ese tratamiento no es el correcto. De alguna manera, se debía incluir la tasa interna de retorno (k), de tal forma que el descuento de los flujos de caja en el tiempo se realice de la forma clásica que todos conocemos. Esto cambia directamente el valor que reporta a la organización la subcartera solución seleccionada.

Por otra parte, la solución que obtiene el algoritmo (y todas las propuestas en la literatura de programación de proyectos) se reduce en último término a un criterio de prioridad: ordenar a todas las actividades según una ley o criterio preestablecido, de tal forma que las vamos incluyendo en el horizonte temporal una a una, conforme sea posible. En el caso original, utilizan el llamado método serie, donde de la lista creada, se van incluyendo una a una en el primer instante de tiempo en el que se pueda (teniendo en cuenta la limitación de recurso global y un tiempo máximo si lo hubiese).

La regla que ellos utilizaban estaba basada simplemente en tiempos (duración de las actividades y su inicio deseado según los precios en los slots en esa iteración). En cambio, nuestra propuesta

pretende involucrar algo básico en el proyecto: su valor. Es evidente que será más interesante priorizar una actividad en el caso de pertener a un proyecto más valioso, para que así ese proyecto pueda terminar lo antes posible.

Nuestro análisis, además de involucrar diferentes niveles para el límite de recurso global, usa diferentes pasos para la actualización de precios en cada iteración y compara las programaciones de proyectos que ambos métodos nos dan (el original – método I – y el que incorpora nuestras modificaciones – método II –).

Cabe añadir que, además del uso de pasos fijos, nosotros proponemos un paso variable en el que, dependiendo de si se supera el recurso disponible en el slot o no, el paso de subida sea uno, y el de bajada sea otro. Es una propuesta que algunos autores han utilizado, y que nos daba la opción de incorporar otra posible mejora a la programación.

En conjunto teníamos 396 carteras diferentes a ejecutar por ambos métodos. La particularidad de la variable respuesta en este análisis (obtenemos tiempos de finalización de proyectos y unidades de beneficio de esa solución) hace que no podamos realizar un análisis de experimentos al uso, y por ello, las conclusiones que obtenemos quedan limitadas a porcentajes de efectividad u ocasiones en las que se obtienen mejores soluciones, lo que por otra parte, no deja de ser la presentación de resultados más orientativa.

En esta disposición, nos proponemos a recordar cuáles fueron las principales conclusiones que se obtuvieron en dicho análisis:

- En más del 90% de las carteras ejecutadas, el resultado obtenido en el método II es igual o mejor que el que proporcionaba el I. Cuando nos referimos a una solución mejor, hablamos de carteras que reportan mayor beneficio en su programación. En principio, la nueva regla de prioridad perseguía este objetivo, luego en lo que a esto respecta, conseguimos una mejora muy importante.

- Sistemáticamente el paso variable en ambos métodos reporta la solución óptima o al menos, programación de subcarteras de mayor valor. En todos los casos, incluso con paso fijo, debemos tener en cuenta que la solución está limitada por el criterio de parada que elegimos, aunque usemos el mismo siempre, luego quedan igual de perjudicados o beneficiados.

- Cuando ambos métodos llegan a la misma programación, el segundo método tiene un uso medio de utilización de recursos frente a los que podría disponer (lo que hemos llamado AUF_{medio}) mayor. Es decir, son soluciones que saturan más los recursos de los que disponen, y por tanto, hacen un uso más efectivo.

Debido a la limitación temporal de la aplicación en Netlogo (la programación por capas y salida visual de agentes, junto con los duros fundamentos matemáticos que resuelven el algoritmo como el método de Push-Relabel para el cálculo de flujo máximo en una red de nodos y arcos, ralentizan la conclusión del programa), tenemos que recurrir a proyectos sencillos, con pocas actividades y con pocas relaciones de precedencia, lo que hace que las carteras que afrontamos no sean demasiado habituales en la vida real de una organización, pero de alguna forma esto nos permite comprobar hasta cierto punto, si las soluciones obtenidas son correctas y factibles, y llegar a una aplicación con un carácter un tanto didáctico.

Por último, básico en cualquier proyecto, nos planteamos su posible comercialización. De ahí, el tema sobre el estudio económico, el cuál pretende recopilar los gastos que ha supuesto la realización de este PFC, y qué precio deberíamos poner a nuestro trabajo haciendo la hipótesis de unas determinadas ventas de la aplicación a PYMES y grandes empresas en los próximos años.

VIII.b. Futuras líneas de investigación

Al igual que este proyecto partía de otro informe precedente en el que se replicaba este algoritmo en lenguaje JAVA, este documento pretende también ser soporte para otros posibles proyectos fin de carrera o al menos, ser una buena referencia en torno a las temáticas que han sido tratadas y el modo de análisis de un algoritmo de estas características.

Como ya comentamos, la limitación de la programación en cuanto al tiempo de ejecución es el principal hándicap de la aplicación. La resolución de cada iteración supone la creación de grafos con tantos nodos como posibles slots de inicio puedan tener todas las actividades, y número de arcos proporcionales a las relaciones de precedencia entre actividades. Resolver el flujo máximo que circula por cada una de estas redes (una para cada proyecto) requiere mucho tiempo de ejecución (método Push-Relabel y posteriormente cálculo del corte mínimo para obtener los inicios de las actividades), por lo que es difícil extrapolar esta programación a proyectos muy complejos, ya que sería inabordable llegar a la solución.

Partiendo de esta realidad, las posibles opciones que se pueden plantear son:

- Habiendo realizado la programación en Netlogo, donde obtenemos como ventajas un planteamiento más visual y didáctico del funcionamiento del algoritmo, replantearse una reprogramación en otro lenguaje, que consiga ser más rápido y que no tenga tan limitadas las salidas gráficas como en el caso de JAVA.

  En ese caso, un posible siguiente paso sería la creación de una librería de carteras para ejecutar. Se crearían diferentes carteras, con proyectos más realistas (podemos encontrar de más de 100 actividades) y en las que se pusieran en juego diferentes variables propias de nuestro problema (horizonte temporal, recursos límite (locales y globales), relaciones de

precedencia, valor máximo del proyecto) de tal forma que se pudiera hacer una batalla de experimentos más compleja de la que se obtuvieran otro tipo de conclusiones. Además, la ventaja de la librería es poder estandarizar las carteras a ejecutar de tal forma que otras heurísticas puedan resolverlas y poder comparar así resultados entre todas ellas.

- Siguiendo con la aplicación en Netlogo y con proyectos más simples, se pueden valorar infinitas reglas de prioridad en el método serie. Además de las mencionadas en el informe, pueden valorarse otras por el propio autor, de tal forma que se puedan comparar muchos más métodos derivados.

  Por otra parte, el futuro autor no se debe verse condicionado únicamente al método serie, puesto que como ya vimos en el capítulo III, hay muchas opciones de heurísticas de asignación de prioridades. Sería interesante replantear la programación con un método paralelo, por ejemplo, y ver si los resultados cambian sustancialmente. En principio es de esperar que así lo sea, puesto que sería modificar aquello que rige la colocación de actividades en la programación.

- Otro aspecto interesante es la valoración de los proyectos en el algoritmo. Si bien, un futuro investigador debería mantener la incorporación que se hizo en este PFC de la tasa de descuento k para los flujos de caja en el tiempo, todos los experimentos se han hecho con los valores máximos de los cuatro proyectos presentes en el artículo (6, 8, 11 y 9 respectivamente).

  Sería interesante modificar estos valores según algún criterio, o incluso valorar a todos por igual, de tal forma que emergieran otras posibles variables que condicionan que un proyecto entre o no en la subcartera solución: duración, complejidad, menor uso de recursos….

- Con respecto a restricciones de recursos, en nuestro informe hemos puesto un único recurso necesario para todos los proyectos (ya que hasta cierto punto podíamos hacer comprobaciones de un correcto funcionamiento del algoritmo gracias a ello), mientras que la programación está preparada para abordar hasta cuatro recursos (con sus limitaciones pertinentes). En próximos análisis se pueden hacer más complejas las carteras jugando con estas variables, lo que consigue una visión algo más realista de los proyectos reales que hay en organizaciones, donde se necesitan varios recursos evidentemente.

- Otro factor condicionante es el paso en la actualización de precios. Nosotros hemos propuesto un paso variable que ha reportado mejores resultados, pero simplemente se daba un paso para la subida de precios, y la mitad de éste en el caso de bajada.

  Por ello, se puede intentar modificar esta regla de cambio de paso de una forma algo más compleja, que quizás consiga hacer converger más rápido al algoritmo. Si no es así, siempre se pueden valorar diferentes propuestas y ver en qué iteración se obtiene la solución.

- Sabiendo lo que el método para obtención de flujo máximo supone para la programación, quizás se podría implementar otra heurística. No deja de ser una de las opciones más arriesgadas y bien es cierto, que en la bibliografía el método Push-Relabel está bien considerado.
  En el caso de plantearse algo así, ya nos alejaríamos de una réplica como tal del algoritmo del artículo, lo que también podría ser interesante, una nueva propuesta.

- Si recordamos, una de las aportaciones ha sido la nueva valoración de los proyectos, incluyendo la tasa de descuento. Siguiendo por este camino, se podría conjugar esta programación con toda la parte de financiación que puede estar detrás de la ejecución de proyectos, de tal forma que las programaciones puedan estar condicionadas por diferentes flujos de caja (positivos, negativos) y que se pueda llegar a desestimar un proyecto porque no se va a poder abordar según esas condiciones previstas.
  Siempre se pretende aproximar lo máximo posible a ese punto de vista más realista que se encuentra en las organizaciones reales.

- Por último, se podría proponer otro criterio de parada para la aplicación. En nuestro caso, se decidió parar tras un número fijo de iteraciones sin encontrar mejor solución.
  Otro posible planteamiento sería involucrar a los valores del límite superior e inferior para cada cartera. En cada instancia, no siempre los límites consiguen converger igual, por lo que no se trataría ya tanto de fijar un porcentaje de proximidad entre ambos límites para todos los casos, si no conseguir particularizar ese gap según las características de la cartera inicial.

Aunque éstas son las líneas más claras que se pueden proponer para quién quiera continuar con el desarrollo de esta aplicación, es cierto que al involucrarse en el funcionamiento del algoritmo y llegando a interiorizar todas sus etapas al afrontar su programación, es muy probable que futuros sucesores puedan ver otras opciones de mejora o simplemente cambios que inquieten su interés.

Por ello, conviene no cerrarse puertas en torno a todas las variantes que algoritmos de este tipo ponen a nuestra disposición.

IX. BIBLIOGRAFÍA

- Álvaro Marcos García, "Algoritmo basado en subastas combinatorias para la optimización de carteras de proyectos" (2012).

- Angela H.L. Chen and Chiuh-Cheng Chyu, "A memetic algorithm for maximizing net present value in resource-constrained project scheduling problem", 2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence) (June 2008): 2396 – 2403.

- Antonio Lova and Pilar Tormos, "Combining Random Sampling and Backward-Forward heuristics for Resource-Constrained Multi-Project Scheduling", in Worshop on Project Management and Scheduling, 2002 (Workshop on Project Management and Scheduling, 2002).

- Barr, RS and Golden, BL, "Designing and Reporting on Computational Experiments with Heuristic Methods" (1995).

- Barr, RS and Golden, BL, "Guidelines for Designing and Reporting on Computational Experiments with Heuristic Methods" (2001).

- Boris V Cherkassky and Andrew V Goldberg, "On implementing the Push-Relabel Method for the Maximum Flow Problem", Algorithmica 19, no. 4 (December 1997): 390-410.

- Christodoulos A. Floudas and Panos M. Pardalos "Encyclopedia of Optimization" Springer Reference (Second Edition).

- Eric L Demeulemeester and Willy S Herroelen, "Project Scheduling: A Research Handbook", ed. Frederick S Hillier, Optimization, vol. 102 (Kluwer Academic Pub, 2002).

- Giuseppe Confessore, Stefano Giordani, and Silvia Rismondo, "A market-based multi-agent system model for decentralized multi-project scheduling", Annals of Operations Research 150, no.1 (December 2006): 115 – 135.

- J Chen and R Askin, "Project selection, scheduling and resource allocation with tiem dependent returns", European Journal of Operational Research 193, no.1 (February 16, 2009): 23 – 34.

- Marshall L Fisher, "The Lagrangian Relaxation Method for Solving Integer Programming Problems", Management Science 50, no. 12 (December 2004): 1861 – 1871.
- Pilar Tormos and Antonio Lova, "A competitive heuristic solution technique for resource-constrained project scheduling", Annals of Operations Research 102 (2001): 65 – 81.

- Rainer Kolisch, "Serial and parallel resource-constrained project scheduling methods revisited: Theory of computation", European Journal of Operational Research 90, no. 2 (April 1996): 320 – 333.

- Rolf H Möhring et al., "Solving Project Scheduling Problems by Minimum Cut Computations", Management Science 49, no. 3 (2003): 330 – 350.

- Yong-yi Shou and Yi-Iun Huang, "Combinatorial auction algorithm for project portfolio selection and scheduling to maximize the net present value". Journal of Zhejiang University SCIENCE C 11, no. 7 (July, 2010): 562 – 574.

ANEXOS:

CÓDIGO EN NETLOGO DEL PROGRAMA

```
__includes ["01. ALMACENAJE PROYECTOS.nls" "02. FECHA MIN - MAX ACTIVIDADES.nls" "03. CREACIÓN GRAFO.nls" "04. PUSH-RELABEL.nls" "05. Funciones PUSH-RELABEL.nls" "06. CORTE MÍNIMO.nls" "07. DEMANDA.nls" "08. GRAFO DEMANDA.nls" "09. LÍMITES.nls" "10. MÉTODO SERIE.nls" "11. ACTUALIZADOR PRECIOS.nls" "12. ITERACIÓN.nls" "13. CORTES.nls" "14. SETUP GRAFO.nls"]

globals [string ruta rutaOrdenador rutaProj vectorRutas numProj lista start recursos cadena
         numAct horizon numSuc sucesoras dura_reque dispo_recur ind duracionActividades
         necesidadesActividades fechasLim indProy flujoMax mayorhorizonte matrizPrecios
         matrizDemanda matrizRecursos recursosUtilizados matrizSerie UB LB diferencia
         carteraInicial carteraMS iteracion solucion mejoraSol check posiblesRutas activos]

breed [ proyectos proyecto ]

breed [ nodos nodo ]

breed [ sellos sello ]

proyectos-own [ id_proyecto N_enCARTERA startProy totalAct horiz cantidadSucesoras actividadesSucesoras
                duracionAct necesidadesAct disponiRecursos fechasLímite NPVMax
                Utilidad opcionCortes actividadesAntecesoras]

nodos-own [ id_proyecto id_act slot duracion fechaMin fechaMax _label id_nodo exceso activo ]

sellos-own [ id_proyecto id_act _inicio slot dura recurso prioridad In/Out instanteFin]

links-own [ id_proyec _slot id_act_ini id_act_fin flujo capacidad residual admisible ]


;;-------------------------------------------------------------------------------------------------

to abrir ;; Sale un cuadro para que decidas ejecutarlo o no


  if user-yes-or-no? "¿Ejecutar nueva cartera?"[

    ca
    file-close-all

    resize-world 0 100 0 30

    let contador 0
    set ruta []
    set rutaProj [0]
    set numProj "0"
    set start []

    set rutaordenador user-directory
    set-current-directory rutaordenador

    file-open user-new-file

    while [file-at-end? = false] [

      set string file-read-line


      if (contador = 3)[

        let separador "\\"
        set ruta remove "<name>" remove "</name>" remove "\t" remove " " string
        let aux substring ruta 8 10
        set aux remove "_" aux
        set numProj aux

        type "Proyectos en cartera >>>> " print numProj
        set aux (word rutaordenador separador)
        set ruta word aux ruta
        set vectorRutas [0]

      ]


      if (contador >= 3 and contador <= 4 + (read-from-string numProj) * 4)[

        let aux1 substring string 5 9


        if (aux1 = "file")[
```

```
          let separador "\\"
          set rutaProj remove "<filename>" remove "</filename>" remove "\t" remove " " string
          set rutaProj replace-item 16 rutaProj "\\"
          set rutaProj replace-item 20 rutaProj "\\"
          let aux word ruta separador
          set rutaProj word aux rutaProj

          set vectorRutas lput rutaProj vectorRutas

        ]

        set aux1 substring string 5 9

        if (contador = 7) [set start [0]]

        if (aux1 = "star")[

          let aux remove "<start>" remove "</start>" remove "\t" remove " " string
          set start lput aux start ;; Vector con las fechas de comienzo de los proyectos

        ]

      ]



      if (contador = 7 + (read-from-string numProj)* 4)[set recursos [0]]



      if (contador >= 7 + (read-from-string numProj)* 4 and contador <= 10 + (read-from-string numProj)* 4)[

        let aux remove "<resource>" remove "</resource>" remove "\t" remove " " string
        set recursos lput aux recursos

      ]


      set contador (contador + 1)

    ] ;; FINAL BUCLE WHILE DE LECTURA DE CADA LINEA DEL FICHERO

    set numProj read-from-string numProj
    set start but-first start
    set recursos but-first recursos
    set vectorRutas but-first vectorRutas

    set start map [read-from-string ?] start

    ;; print vectorRutas
    ;; type "Comienzo de cada proyecto >>>> " print start
    ;; type "Restricción de recursos >>>> " print recursos


    file-close

    explorar numProj vectorRutas start recursos

  ] ;; FINAL DEL BUCLE DE TODO EL PROCEDIMIENTO. Dando al "SI" cargamos el .xml que escojamos

end


;;-----------------------------------------------------------------------------------------

to explorar [ NUMERO_PROYECTOS VECTOR_DE_RUTAS COMIENZO_PROYECTOS REC_GLOBAL]

  ca

  let i 0

  while [i < (length VECTOR_DE_RUTAS)][

    let contador 0
    set numAct "200"


    file-open item i VECTOR_DE_RUTAS
```

```
while [ file-at-end? = false ][

  set cadena []
  set cadena file-read-line

  if (contador = 5)[let aux substring cadena 33 35 set numAct word aux numAct
                   set numAct remove "200" numAct]


  if (contador = 6)[let aux substring cadena 33 36 set horizon[0] set horizon lput aux horizon]


  if (contador = 18)[set numSuc [0] set sucesoras [0]]


  if ((contador >= 18) and (contador < read-from-string numAct + 18))[

    set numSuc lput (item 23 cadena) numSuc

    if (item 23 cadena = "0" )[ set sucesoras sentence sucesoras [[0 0 0]]]

    ;; Introduzca tres ceros como sucesoras "no existentes"

    if (item 23 cadena = "1" )[

      let aux substring cadena 34 36
      set aux read-from-string aux
      set aux sentence aux [0 0]
      ;; Para estandarizar el vector de sucesoras y que cada actividad tenga un vector
      ;; asociado de dimensión 3
      set sucesoras lput aux sucesoras

    ]

    if (item 23 cadena = "2" )[

      let aux1 substring cadena 34 36
      set aux1 read-from-string aux1
      let aux2 substring cadena 38 40
      set aux2 read-from-string aux2
      let aux sentence aux1 aux2
      set aux sentence aux [0]
      set sucesoras lput aux sucesoras

    ]

    if (item 23 cadena = "3")[

      let aux1 substring cadena 34 36
      set aux1 read-from-string aux1
      let aux2 substring cadena 38 40
      set aux2 read-from-string aux2
      let aux3 substring cadena 42 44
      set aux3 read-from-string aux3
      let aux sentence aux1 aux2
      set aux sentence aux aux3
      set sucesoras lput aux sucesoras
    ]

  ]


  let fila read-from-string numAct

  if (contador = fila + 22) [set dura_reque [0]]


  if ((contador >= fila + 22) and (contador < fila * 2 + 22))[

    let aux substring cadena 14 16
    set dura_reque lput aux dura_reque
    set aux substring cadena 22 24
    set dura_reque lput aux dura_reque
    set aux substring cadena 27 29
    set dura_reque lput aux dura_reque
    set aux substring cadena 32 34
    set dura_reque lput aux dura_reque
    set aux substring cadena 37 39
    set dura_reque lput aux dura_reque
```

```
      ]

      ;; Consigo un vector fila con 5 items para cada actividad,
      ;; en orden: duración + necesidad R1 + necesidad R2 + ...R3 + ...R4


      if (contador = fila * 2 + 25 ) [

        set dispo_recur [0]
        let aux substring cadena 3 5
        set dispo_recur lput aux dispo_recur
        set aux substring cadena 8 10
        set dispo_recur lput aux dispo_recur
        set aux substring cadena 13 15
        set dispo_recur lput aux dispo_recur
        set aux substring cadena 18 20
        set dispo_recur lput aux dispo_recur

      ]

      set contador (contador + 1)


    ] ;; FINAL DEL BUCLE WHILE DE LECTURA DEL FICHERO


    set numSuc but-first numSuc
    set sucesoras but-first sucesoras
    set dura_reque but-first dura_reque
    set dispo_recur but-first dispo_recur
    set horizon but-first horizon


    set horizon map[read-from-string ?] horizon

    ;; HACER QUE LOS VECTORES QUE INICIALMENTE GUARDARON CADENAS DE TEXTO, SE CONVIERTAN EN NÚMEROS

    set numSuc map [read-from-string ?] numSuc
    set dura_reque map [read-from-string ?] dura_reque
    set dispo_recur map [read-from-string ?] dispo_recur


    file-close

    set i (i + 1)

    almacenaje NUMERO_PROYECTOS COMIENZO_PROYECTOS numAct horizon numSuc sucesoras dura_reque  dispo_recur
REC_GLOBAL

    ;; "almacenaje" guarda los datos de cada proyecto en agentes "proyecto"

  ]

end
```

01. ALMACENAJE PROYECTOS

```
to almacenaje [ numeroProyCartera startProyectos numeroAct horizonte numeroSucesoras vectorSucesoras
Durac_Necesid Disponib_Recursos Recursos_glob]

   set-default-shape proyectos "square"

   let j 0

   if (item 0 horizonte > mayorhorizonte) [

     ;; Inicializar la matriz de precios y de recursos globales.
     ;; 4 valores para los 4 recursos en cada slot del horizonte.

     set mayorhorizonte item 0 horizonte

     set matrizPrecios [] set matrizRecursos []

     while [j < mayorhorizonte ][

       set matrizPrecios lput [0 0 0 0] matrizPrecios
       set matrizRecursos lput Recursos_glob matrizRecursos

       set j (j + 1)
     ]
   ]


   set duracionActividades [0]
   set necesidadesActividades [0]

   set j 0

     while [j < read-from-string numeroAct][

       let aux item (5 * j) Durac_Necesid

       set duracionActividades lput aux duracionActividades

       set aux sublist Durac_Necesid (j * 5 + 1) (j * 5 + 5)

       set necesidadesActividades lput aux necesidadesActividades

      ;; Obtenemos un vector de vectores con tantos items como actividades

       set j (j + 1)

     ]

     set duracionActividades but-first duracionActividades
     set necesidadesActividades but-first necesidadesActividades


     create-proyectos 1 [set id_proyecto count proyectos]

     ask proyectos with [id_proyecto = (ind + 1)][set NPVMax read-from-string user-input
                                                   "¿NPV máximo del proyecto?"]

     ;; Ahora asignamos las propiedades a cada agente (proyecto) de la cartera

     ask proyectos with [id_proyecto = (ind + 1) ][

       set heading 90
       set xcor 0.5 set ycor (10 + 2 * ind)
       set size 2
       set label (word (ind + 1) "   ")
       set label-color 50
       set color (15 + ind * 10)

     ]

     ask proyectos with [id_proyecto = (ind + 1) ][set N_enCARTERA numeroProyCartera]
     ask proyectos with [id_proyecto = (ind + 1) ][set startProy item ind startProyectos]
     ask proyectos with [id_proyecto = (ind + 1) ][set totalAct read-from-string numeroAct]
     ask proyectos with [id_proyecto = (ind + 1) ][set horiz item 0 horizonte]
     ask proyectos with [id_proyecto = (ind + 1) ][set actividadesSucesoras vectorSucesoras]
     ask proyectos with [id_proyecto = (ind + 1) ][set duracionAct duracionActividades]
     ask proyectos with [id_proyecto = (ind + 1) ][set necesidadesAct necesidadesActividades]
     ask proyectos with [id_proyecto = (ind + 1) ][set disponiRecursos Disponib_Recursos]
```

```
    set fechasLim
   (fechaMinMax read-from-string numeroAct vectorSucesoras duracionActividades item 0 horizonte)

    ask proyectos with [id_proyecto = (ind + 1) ] [set fechasLímite fechasLim ]
    ask proyectos with [id_proyecto = (ind + 1)] [set Utilidad NPVMax]

    crearGrafo (read-from-string numeroAct)
   (item 0 horizonte) duracionActividades necesidadesActividades vectorSucesoras fechasLim



    ask proyectos with [id_proyecto = (ind + 1) ][set hidden? true]

    set ind (ind + 1)


    ;; Si ha entrado en este procedimiento tantas veces como proyectos tiene la cartera,
    ;; entonces se han registrado todos correctamente

    if (numeroProyCartera = ind) [

      print "Registro de la cartera CORRECTO"
      show date-and-time

      set solucion []
      set diferencia 0 ;; Para inicializar la diferencia entre UB y LB
      set mejoraSol []

      let i 0

      while [i < numeroProyCartera][

        ask proyectos with [id_proyecto = (i + 1) ][set hidden? false]

        let aux []
        set j 1

        while [j <= item 0 [totalAct] of proyectos with [id_proyecto = (i + 1)] - 2][

          set aux lput item 0 item j item 0 [fechasLímite] of proyectos with
                              [id_proyecto = (i + 1)] aux

          set j (j + 1)
        ]

        schedule i aux

        set i (i + 1)

      ]

      iterar

    ]

end
```

02. FECHA MIN-MAX ACTIVIDADES

```
to-report fechaMinMax [totalActividades VecSucesoras duraciones horizonte]

  let i 0
  let j 0
  let k 0
  let m 0
  let fmin 0
  let predecesora 0
  let aux1 0
  let aux2 0
  let aux 0

  let vectorfechas n-values totalActividades [[0 500]]
  ;; Inicializamos el vector con tantos items como actividades, con fechaMin 0 y Max de 500

  ;; CÁLCULO DE FECHAS MÍNIMAS

  while [ i <= (totalActividades - 1)][ ;; del item 0 al item (totalActividades - 1)


    while [ j <= (totalActividades - 1)][

      set k 0

      while [ k < 3][ ;; Los items del vector de sucesoras van de tres en tres

        set predecesora (item k item j VecSucesoras)

        if (predecesora != 0)[
        ;; Si en el vector de sucesoras hay un 0, no hay sucesora, luego que no haga nada

          set m (predecesora - 1)
          ;; El índice de la actividad predecesora corresponde a un item en vectorfechas en la
          ;; posición anterior


          set aux1 (item j duraciones)
          set aux2 (item 0 item j vectorfechas)
          set aux (aux1 + aux2)


          if (aux > item 0 item m vectorfechas)[

            set fmin aux

            set vectorfechas (replace-item m vectorfechas (replace-item 0 (item m vectorfechas) fmin))
            ;; Sustituir fmin donde corresponde

           ]
        ]

        set k (k + 1)

     ]

    set j (j + 1)

    ]

  set i (i + 1)

  ]



  ;; CÁLCULO DE FECHAS MÁXIMAS

  set i (totalActividades - 2) ;; Inicializamos de nuevo los contadores
  let sucesora 0
  let fmax 0

  set vectorfechas (replace-item (totalActividades - 1) vectorfechas
      (replace-item 1 (item (totalActividades - 1) vectorfechas) horizonte))

  ;; La fecha máxima de la última actividad será el horizonte del proyecto
```

```
 while [ i >= 0 ][

    set k 0

    while [ k < 3] [

      set sucesora (item k item i VecSucesoras)

      if ( sucesora != 0 )[

        set m (sucesora - 1)

        set aux1 (item 1 item m vectorfechas)
        set aux2 (item i duraciones)

        set aux (aux1 - aux2)

        if (aux < item 1 item i vectorfechas)[

          set fmax aux

          set vectorfechas (replace-item i vectorfechas (replace-item 1 (item i vectorfechas) fmax))
          ;; Sustituir fmax donde corresponde

        ]

      ]


    set k (k + 1)

    ]


  set i (i - 1)  ;; Desde la penúltima actividad hasta la primera

  ]

  report vectorfechas

end
```

03. CREACIÓN GRAFO

```
to crearGrafo [numActividad horizont duracActividad necesidades suceActividad fechasLimi]

     let i 1
     let j 0
     let k 0

     set-default-shape nodos "circle"

     create-nodos 1 [set size 1.5 set xcor 5 set ycor max-pycor / 2 set label "S  "
     set id_proyecto indProy set color 15 + indProy * 10 set id_nodo 1] ;; Nodo fuente (s)

     create-nodos 1 [set size 1.5 set xcor 3 * horizont + 10 set ycor max-pycor / 2 set label "T  "
     set id_proyecto indProy set color 15 + indProy * 10 set _label 0] ;; Nodo sumidero (t)


;; Creación de los NODOS del grafo

       while [i <= (numActividad - 2)][

         set j 0

         while [j <= horizont][


           if (j >= item 0 item i fechasLimi and j <= item 1 item i fechasLimi + 1) [

             create-nodos 1 [ set size 0.5 set id_nodo (count nodos) set id_act (i + 1)
                              set slot j set id_proyecto (count proyectos - 1)]

             ask nodos with [id_act = (i + 1) and slot = j and id_proyecto = indProy][

               set heading 90 set xcor (10 + 3 * j)
               set ycor (max-pycor - 2) - (max-pycor / numActividad) * i
               set color (15 + 10 * indProy)
               set duracion (item i duracActividad)
               set fechaMin (item 0 item i fechasLimi)
               set fechaMax (item 1 item i fechasLimi)
               set _label 0
               set exceso 0
               set activo 0
             ]

           ]

           set j (j + 1)
         ]


;; Creación de los ARCOS del grafo

         set j 0

         while [j <= horizont][

           if (j = item 0 item i fechasLimi)[              ;; Desde la fuente S

             ask nodos with [label = "S  " and id_proyecto = indProy][

               create-links-to nodos with [id_proyecto = indProy and
                                           id_act = (i + 1) and slot = j][

                 set id_proyec indProy
                 set capacidad 10000
                 set residual (capacidad - flujo)
                 set admisible 1
               ]
             ]

             ask nodos with [id_proyecto = indProy and id_act = (i + 1) and slot = j][

               create-links-to nodos with [label = "S  " and id_proyecto = indProy][

               set id_proyec indProy
               set capacidad 0
               set residual (capacidad - flujo)
```

```
    set admisible 1

    ]

  ]

]


if (j = item 1 item i fechasLimi + 1)[           ;; Hasta el sumidero T

  ask nodos with [id_proyecto = indProy and id_act = (i + 1) and slot = j][

    create-links-to nodos with [label = "T  " and id_proyecto = indProy][

      set id_proyec indProy
      set capacidad 10000
      set residual (capacidad - flujo)
      set admisible 1

    ]
  ]

  ask nodos with [label = "T  " and id_proyecto = indProy][

    create-links-to nodos with [id_proyecto = indProy and id_act = (i + 1) and slot = j][

      set id_proyec indProy
      set capacidad 0
      set residual (capacidad - flujo)
      set admisible 0

    ]

  ]

]

;; Arcos de asignación

if (j >= item 0 item i fechasLimi and j <= item 1 item i fechasLimi + 1)[

  ask nodos with [id_proyecto = indProy and id_act = (i + 1) and slot = j][

    create-links-to nodos with [id_proyecto = indProy and id_act = (i + 1)
                                and slot = (j + 1)][

      let h j
      set id_proyec indProy
      set _slot j
      set id_act_ini (i + 1)
      set id_act_fin (i + 1)
      set admisible 0

     while [h < j + item 0 [duracion] of nodos with [id_proyecto = indProy and
                                                     id_act = (i + 1) and slot = j]][

        set k 0

        while [k < 4][

          set capacidad precision (capacidad + item k item h matrizPrecios
                                   * item k item i necesidades) 5
          set residual precision (capacidad - flujo) 5

          set k ( k + 1)
        ]

       set h (h + 1)
      ]

     if ( i = (numActividad - 2) ) [

       set capacidad precision (capacidad + 0.05 *
                                item 0 [NPVMax] of proyectos with [id_proyecto = indProy + 1] *
                               ( j - item 0 item i item 0 [fechaslÃ-mite] of proyectos with
                                [id_proyecto = indProy + 1]))5
       set residual precision (capacidad - flujo) 5
     ]

    ]
```

```
      ]

      ask nodos with [id_proyecto = indProy and id_act = (i + 1) and slot = (j + 1)][

        create-links-to nodos with [id_proyecto = indProy and id_act = (i + 1) and slot = j][

          set id_proyec indProy
          set id_act_ini (i + 1)
          set id_act_fin 0
          set _slot j
          set capacidad 0
          set flujo 0
          set admisible 0

          set residual precision (capacidad - flujo) 5

        ]

      ]

    ]

   set j (j + 1)
  ]


set i (i + 1)
]



set i 0
let actSuc 0
let limite 0


while [i <= (numActividad - 2)][

  set k 0

  while [k < 3][

    set actSuc item k item i suceActividad

    let m 0

    while [ m <= horizont ][

    set limite (m - item 0 item 1 fechasLimi) + (item i duracActividad - 1)

    set j limite

    while [ j <= horizont and actSuc != 0 and actSuc != numActividad][

      if (j >= item 0 item i fechasLimi - 1
          and j <= item 1 item i fechasLimi
          and (j + item i duracActividad) >= item 0 item (actSuc - 1) fechasLimi
          and (j + item i duracActividad) <= item 1 item (actSuc - 1) fechasLimi)[

        ;; Arcos temporales

        ask nodos with [id_proyecto = indProy and id_act = (i + 1) and slot = j][

          create-links-to nodos with [id_proyecto = indProy and id_act = actSuc and
                                      slot = (j + item i duracActividad)][

            set id_proyec indProy
            set id_act_ini (i + 1)
            set id_act_fin actSuc
            set capacidad 10000
            set residual precision (capacidad - flujo) 5
            set admisible 0

          ]

        ]

        ask nodos with [id_proyecto = indProy and id_act = actSuc and
                        slot = (j + item i duracActividad)][
```

```
            create-links-to nodos with [id_proyecto = indProy and id_act = (i + 1) and slot = j][

              set id_proyec indProy
              set id_act_ini actSuc
              set id_act_fin (i + 1)
              set capacidad 0
              set residual precision (capacidad - flujo) 5
              set admisible 0

            ]

          ]

        ]

        set j (j + 1)
      ]

        set m (m + 1)
      ]

      set k (k + 1)
    ]

  set i (i + 1)
]


let origen item 0 [who] of nodos with [id_proyecto = indProy and label = "S  "]

ask nodos with [label = "S  " and id_proyecto = indProy]
               [set _label count nodos with [id_proyecto = indProy]]
ask nodos with [label = "S  " and id_proyecto = indProy]
               [set exceso sum [capacidad] of links with
                [id_proyec = indProy and end1 = (nodo origen)]]

set mayorhorizonte horizont


set i 1

let actAntecesoras []
repeat (numActividad - 3) [set actAntecesoras lput [] actAntecesoras ]

;; Cálculo de antecesoras de cada actividad. Útil para crear los posibles cortes del grafo válidos

while [i < (numActividad - 2)][

  let m 0

  while [m < 3][

   if (item m item i item 0 [actividadesSucesoras] of proyectos with [id_proyecto = indProy + 1] != 0)[

     let aux (i + 1)
     let aux2 (item m item i item 0 [actividadesSucesoras] of proyectos with
               [id_proyecto = indProy + 1] - 3)

     set actAntecesoras replace-item aux2 actAntecesoras (lput aux (item aux2 actAntecesoras))

   ]

   set m (m + 1)
  ]


  set i (i + 1)
]


ask proyectos with [id_proyecto = indProy + 1][set actividadesAntecesoras actAntecesoras]



;; Cálculo de los n-cuts dentro del grafo, así sólo lo calcularemos en la primera iteración

set i 1
let aux []
set posiblesRutas []

set j item 0 item i item 0 [fechasLí-mite] of proyectos with [id_proyecto = indProy + 1]
```

```
    while [ j <= item 1 item i item 0 [fechasLí-mite] of proyectos with [id_proyecto = indProy + 1] ][

      set aux []
      set aux lput j aux

      posiblesCortes indProy aux i j

      set j (j + 1)
    ]


    ask proyectos with [id_proyecto = indProy + 1][set opcionCortes posiblesRutas]



    set actAntecesoras fput [0] actAntecesoras
    ask proyectos with [id_proyecto = indProy + 1][set actividadesAntecesoras actAntecesoras]

    ask nodos with [id_proyecto = indProy][set hidden? true]
    ask links with [id_proyec = indProy][set hidden? true]


    set indProy ( indProy + 1 )


end
```

04. PUSH-RELABEL

```
to PushRelabel [ID]

  print "Push Relabel... "
  let fuente 0
  let nodoFu 0
  let sumidero 0
  let nodoSu 0

  if (ID = 0)[set flujoMax []] ;; En una nueva iteración, inicializo el vector de flujo máximo

  ask nodos with [id_proyecto = ID][set hidden? false]
  ask links with [id_proyec = ID][set hidden? false]

  set fuente item 0 [who] of nodos with [id_proyecto = ID and label = "S  "]
  set sumidero item 0 [who] of nodos with [id_proyecto = ID and label = "T  "]


  set nodoFu sort-by < [end2] of links with [id_proyec = ID and end1 = (nodo fuente)]
  ;; Nodos unidos a la fuente

  set nodoSu sort-by < [end1] of links with [id_proyec = ID and end2 = (nodo sumidero)]
  ;; Nodos unidos al sumidero

  ask nodo fuente[

    set exceso sum [capacidad] of links with [id_proyec = ID and end1 != (nodo fuente) and
                                              end2 != (nodo fuente) and end1 != (nodo sumidero) and
                                              end2 != (nodo sumidero) and
                                              id_act_ini = id_act_fin ]

    set activo 1
    set _label count nodos with [id_proyecto = ID]
  ]
  ;; Para inicializar el exceso que puede tener la fuente, se puede usar la suma de capacidades
  ;; de toda la red, por ejemplo.

  ask nodo sumidero[

    set activo 0
    set exceso 0
    set _label 0
  ]



  foreach nodoFu [

    ask links with [id_proyec = ID and end1 = (nodo fuente) and end2 = ?][

      set capacidad precision 10000 5
      set flujo 0
      set residual precision (capacidad - flujo) 5
      set admisible 1

    ]

    ask links with [id_proyec = ID and end2 = (nodo fuente) and end1 = ?][

      set capacidad precision 10000 5
      set flujo 0
      set residual precision (capacidad - flujo) 5
      set admisible 0

     ]
  ]

  foreach nodoSu [

    ask links with [id_proyec = ID and end1 = ? and end2 = (nodo sumidero)][

      set capacidad precision 10000 5
      set flujo 0
      set residual precision (capacidad - flujo) 5
      set admisible 1
```

```
    ]

    ask links with [id_proyec = ID and end1 = (nodo sumidero) and end2 = ?][

      set capacidad precision 10000 5
      set flujo (- precision item 0 [flujo] of links with [id_proyec = ID and end1 = ? and
                                                           end2 = (nodo sumidero)] 5)
      set residual precision (capacidad - flujo) 5
      set admisible 0 ;; No podrá retroceder flujo desde el sumidero

     ]
  ]

  set activos []

  push (fuente) ID


  let flujoM precision (item 0 [exceso] of nodos with [id_proyecto = ID and label = "T  "])5

  set flujoMax lput flujoM flujoMax
  ;; Introduzco el valor del flujo máximo de cada proyecto en el vector

  if (Comentarios = true) [print (word "Flujo máximo por la red    " flujoM)]

  ask proyectos with [id_proyecto = ID + 1][set Utilidad NPVMax - flujoM]
  ;; Utilidad del proyecto en esta iteración. Si es positiva pasará a la cartera.

  ask nodos with [id_proyecto = ID] [set hidden? true]
  ask links with [id_proyec = ID] [set hidden? true]

end
```

```
to relabel [u opciones ID]

  let vectorlabel []


  if ([label] of nodo u != "S  ")[

    foreach opciones[

      if (item 0 [residual] of links with [id_proyec = ID and end1 = nodo u and end2 = ?] > 0)[

        set vectorlabel lput ([_label] of ? + 1) vectorlabel]

    ]

    ask nodo u [set _label min vectorlabel]

  ]

  if (activos != [])[

    set activos remove (u) activos
    set activos fput (u) activos

  ]

end


to push [u ID]

  let opciones sort-by < [end2] of links with [id_proyec = ID and end1 = (nodo u)]


  let testigo []

;; ____________________________________________________________________________________
;; ___________________________________________ Si hacemos Push desde la fuente S ____________
;; ____________________________________________________________________________________

if ([label] of nodo u = "S  ")[

    ask nodo u [set color grey]
    let aux [exceso] of nodo u

    foreach opciones [

      let send item 0 [residual] of links with [id_proyec = ID and end1 = ? and
                                                end2 = nodo ([who] of ? + 1)]

      ask links with [id_proyec = ID and end1 = (nodo u) and end2 = ?][

        set flujo precision (flujo + send) 5
        set residual precision (capacidad - flujo) 5
        set color blue

        if ( residual = 0 )[set admisible 0 set thickness 0]

      ]

      ask links with [id_proyec = ID and end1 = ? and end2 = (nodo u)][

        set flujo precision (flujo - send) 5
        set residual precision (capacidad - flujo) 5

      ]

      ask nodo u [set exceso precision (exceso - send) 5]
      ask ? [set exceso precision (exceso + send) 5

       if ([label] of ? != "T  ")
           [set activo 1 set activos lput ([who] of ?) activos set size 1]]

    ]
```

```
]
if ( activos != [])[

;; ____________________________________________ Si hacemos Push desde un nodo estándar _____________
;; ____________________________________________________________________________________________

if ([label] of nodo u != "S  ")[

    ask nodo u [set color green]
    let test 0

    foreach opciones [

      if (item 0 [residual] of links with [id_proyec = ID and end1 = nodo u and end2 = ?] != 0)[set test 1]
      ;; Saber si alguno de sus arcos no está saturado

    ]



    if (test = 1)[ ;; El nodo u tiene algún arco residual

      foreach opciones [

        ifelse ([_label] of nodo u = [_label] of ? + 1)[

          set testigo lput 1 testigo

          if (item 0 [residual] of links with [id_proyec = ID and end1 = nodo u and end2 = ?] > 0)[

            ask links with [id_proyec = ID and end1 = (nodo u) and end2 = ?]
                           [set admisible 1 set thickness 0.2]

          ]

        ][ if ([_label] of ? != "S  ") [ask links with [id_proyec = ID and end1 = (nodo u) and end2 = ?]
                                        [set admisible 0 set thickness 0]]]

      ]

      if ([activo] of nodo u = 1 and testigo = [] ) [relabel u opciones ID]
      ;; El nodo u tiene exceso, pero no arcos admisibles => Relabel



      foreach opciones [

         if ([admisible] of links with [id_proyec = ID and end1 = (nodo u) and end2 = ?] = [1])
             and (item 0 [residual] of links with [id_proyec = ID and end1 = (nodo u) and end2 = ?] > 0)
             and ([label] of ? != "S  ") and ([activo] of nodo u = 1) [

            let send2 0
            let aux ([exceso] of nodo u)
            let aux1 (item 0 [residual] of links with [id_proyec = ID and end1 = (nodo u) and end2 = ?])

            if (aux < aux1)[set send2 aux]
            if (aux1 <= aux)[set send2 aux1]


            ask links with [id_proyec = ID and end1 = (nodo u) and end2 = ?][

              set color blue
              set flujo precision (flujo + send2) 5
              set residual precision (capacidad - flujo) 5

              if (residual = 0) or ([_label] of nodo u != [_label] of ? + 1)
                 [set admisible 0 set thickness 0]
              if (residual > 0) and ([_label] of nodo u = [_label] of ? + 1)
                 [set admisible 1 set thickness 0.2]
            ]

            ask links with [id_proyec = ID and end1 = ? and end2 = (nodo u)][

              set flujo precision (flujo - send2) 5
              set residual precision (capacidad - flujo) 5

              if (residual = 0) or ([_label] of ? != [_label] of nodo u + 1)[set admisible 0 set thickness 0]
              if (residual > 0) and ([_label] of ? = [_label] of nodo u + 1)
                 [set admisible 1 set thickness 0.2]
            ]
```

```
            ask nodo u [

              set exceso precision (exceso - send2) 5

              if ([exceso] of nodo u = 0)[

                set activo 0
                if (activos != [])[set activos remove (u) activos]
                set size 0.5
              ]

            ]

            ask ? [

              set exceso precision (exceso + send2) 5
              if ([exceso] of ? > 0) and ([label] of ? != "T  ")[set activo 1 set size 1
                                                                 set activos fput ([who] of ?) activos]

            ] ;; El nodo que recibe el flujo se activa

         ]


        if ([admisible] of links with [id_proyec = ID and end1 = (nodo u) and end2 = ?] = [1]) and
           ([label] of ? = "S  ")[

          let send [exceso] of nodo u
          ask links with [id_proyec = ID and end1 = nodo u and end2 = ?][set color blue]
          ask nodo u [

            set exceso precision (exceso - send) 5
            set _label ([_label] of ? + 1)
            set activo 0
            if (activos != [])[set activos remove (u) activos]
          ]

          ask ? [set exceso precision (exceso + send) 5]

          if( activos != [])[ push (item 0 activos) ID ]

        ]


       if ([activo] of nodo u = 1) [ relabel u opciones ID ]

    ]
 ]

]

  if ( activos != []) [push (item 0 activos) ID]

]


end
```

06. CORTE MÍNIMO

```
to corteMinimo [ID]

  let corte 10000
  let corteMin 0
  let vector []

  print "Cálculo del corte mí-nimo..."
  ask nodos with [id_proyecto = ID] [set hidden? false]
  ask links with [id_proyec = ID] [set hidden? false]


  let j 0

  while [ j < length (item 0 [opcionCortes] of proyectos with [id_proyecto = ID + 1]) and
         corte != item ID flujoMax][

    let h 0
    set corte precision 0 5

    while [h < item 0 [totalAct] of proyectos with [id_proyecto = ID + 1] - 2 ][

      set corte corte + precision (item 0 [capacidad] of links with
        [id_proyec = ID and id_act_ini = (h + 2) and id_act_fin = (h + 2) and
        _slot = item h item j item 0 [opcionCortes] of proyectos with [id_proyecto = ID + 1]]) 5

      set h (h + 1)
    ]

    set corte precision corte 5
    set vector lput (precision corte 5) vector

    set j (j + 1)

  ]

  if (corte = item ID flujoMax) [

    if ( Comentarios = true) [print (word "MÃ-nima diferencia " precision (min vector - item ID flujoMax) 5 )]

    set corteMin item (j - 1) item 0 [opcionCortes] of proyectos with [id_proyecto = ID + 1]

    print (word " Corte >> " corte " Las actividades del proyecto " (ID + 1) " empiezan en " corteMin)

    schedule ID corteMin

  ]

  if (corte != item ID flujoMax) [ set flujoMax replace-item ID flujoMax (min vector)  corteMinimo2 ID]


end


to corteMinimo2 [ID]

 let j 0
 let corte 10000
 let corteMin 0
 let vector []

  while [ j < length (item 0 [opcionCortes] of proyectos with [id_proyecto = ID + 1]) and corte != item ID
flujoMax][

    let h 0
    set corte precision 0 5

    while [h < item 0 [totalAct] of proyectos with [id_proyecto = ID + 1] - 2 ][

      set corte corte + precision (item 0 [capacidad] of links with [id_proyec = ID and id_act_ini = (h + 2)
       and id_act_fin = (h + 2) and _slot = item h item j item 0 [opcionCortes] of proyectos with
      [id_proyecto = ID + 1]]) 5

     set h (h + 1)
    ]
```

```
    set corte precision corte 5
    set vector lput (precision corte 5) vector

    set j (j + 1)

  ]


    set corteMin item (j - 1) item 0 [opcionCortes] of proyectos with [id_proyecto = ID + 1]

    ask proyectos with [id_proyecto = ID + 1][set Utilidad (NPVMax - item ID flujoMax)]

    if(Comentarios = true) [print (word "MÃ-nima diferencia " precision (min vector - item ID flujoMax) 5 )]

    print (word " Corte >> " corte " Las actividades del proyecto " (ID + 1) " empiezan en " corteMin)

    schedule ID corteMin

end



to schedule [ID inicios]

  set-default-shape sellos "square"

  let i 1
  let j 0

  while [ i <= item 0 [totalact] of proyectos with [id_proyecto = ID + 1] - 2] [

     set j 0

     while [j <= item i item 0 [duracionact] of proyectos with [id_proyecto = ID + 1] - 1][

       create-sellos 1 [

         set heading 90
         set xcor 10 + item (i - 1) inicios + j
         set ycor (max-pycor - 2) -
             (max-pycor / item 0 [totalact] of proyectos with [id_proyecto = ID + 1]) * i
         set color item 0 [color] of proyectos with [id_proyecto = ID + 1]
         set label-color 50
         set size 1.05

         set id_act i
         set id_proyecto ID
         if (i = 1)[set In/Out "Out"]
         if (j = 0)[set _inicio item (i - 1) inicios]
         ;; La primera actividad siempre tendrá la posibilidad de ser programada en el método Serie
         if (j != 0)[set _inicio " "]
         set slot (item (i - 1) inicios) + j
         set dura item i item 0 [duracionact] of proyectos with [id_proyecto = ID + 1]
         set recurso item i item 0 [necesidadesact] of proyectos with [id_proyecto = ID + 1]
         if (j = 0) [set prioridad
         _inicio + 0.5 * (item i item 0 [duracionact] of proyectos with [id_proyecto = ID + 1])]

         ;; MÉTODO 2:
         ;; if (j = 0) [set prioridad _inicio / (item 1 item (item 0 [totalact] of proyectos with
         ;; [id_proyecto = ID + 1] - 1) item 0 [fechasLÃmite] of proyectos with [id_proyecto = ID + 1])]

         if (j != 0)[set prioridad " "]

       ]

       set j (j + 1)
     ]

     ask sellos with [id_act = i and id_proyecto = ID][set label slot + 1]

    set i (i + 1)

  ]

  ask nodos with [id_proyecto = ID][set hidden? true]
  ask links with [id_proyec = ID][set hidden? true]
  ask sellos with [id_proyecto = ID][set hidden? true]


  let cartera []
```

```
  if ( item 0 [N_enCARTERA] of proyectos = ID + 1) [

    set cartera sort-by < [id_proyecto] of proyectos with [Utilidad > 0]

    set cartera map [? - 1] cartera

    set carteraInicial []
    set carteraInicial cartera

    print (word "____Proyectos con utilidad positiva >> " cartera)

    calculoDemanda cartera

  ]

end
```

07. DEMANDA

```
to CalculoDemanda [cartera]

  let horizonte []

  foreach cartera [ ;; Cálculo del horizonte máximo de la subcartera elegida

    set horizonte lput item 0 [horiz] of proyectos with [id_proyecto = ? + 1] horizonte

  ]

  set horizonte max horizonte


  let j 0
  let aux1 0
  let aux2 0
  let vector []
  set matrizDemanda []


  while [j < horizonte][

    let k 0
    set vector []

    while [k < 4][

      set aux1 0
      set aux2 0

      foreach cartera [

        if (any? sellos with [slot = j and id_proyecto = ?])[

          let i 0
          set aux2 0
          let aux [recurso] of sellos with [slot = j and id_proyecto = ?]

          while [i < length [recurso] of sellos with [slot = j and id_proyecto = ?]][

            let num item k item i aux

            set aux2 aux2 + num ;; Obtengo las unidades de recurso k en el slot j para el proyecto ?

            set i (i + 1)
          ]

          set aux1 aux1 + aux2 ;; Acumulo las unidades de recurso k en el slot j de la cartera seleccionada

        ]

      ]

      set vector lput aux1 vector
      ;; Introduzco aux1 para los 4 recursos obteniendo un vector para cada instante j

      set k (k + 1)
    ]

    set matrizDemanda lput vector matrizDemanda


    set j (j + 1)
  ]


  grafoDemanda cartera

  set UB upperBound cartera

  metodoSerie cartera

end
```

```
to-report comienzo [ cotaX ]

  let y 10
  let j 0

  while [j < 30][

    if ( item 0 [pcolor] of patches with [ pxcor = cotaX and pycor = y + j ] = 0 )[

       report (j)

    ]

   set j (j + 1)
  ]


end


to grafoDemanda [cartera]

  ask patches [set pcolor 0 set plabel ""]

  foreach cartera [

    let totalsellos sort-by < [who] of sellos with [id_proyecto = ?]
    let proycolor item 0 [color] of proyectos with [id_proyecto = ? + 1]

    ask proyectos with [id_proyecto = ? + 1][set hidden? false]

    foreach totalsellos [

      let x 10
      let y 10 + comienzo (x + item 0 [slot] of sellos with [who = ?])
      let j 0

      while [j < item 0 item 0 [recurso] of sellos with [who = ?]][

        ask patches with [pxcor = x + item 0 [slot] of sellos with [who = ?] and pycor = y + j]
                         [set pcolor proycolor set plabel [id_act] of sello ?]
        ask patches with [pxcor = 9 and pycor = y + j][set plabel y - 10 + j + 1]

        set j (j + 1)
      ]


      ask patches with [pxcor = x + item 0 [slot] of sellos with [who = ?] and pycor = 9]
                       [set plabel item 0 [slot] of sellos with [who = ?] + 1]
  ]

  ]


end
```

09. LÍMITES

```
to-report upperBound [cartera]

  let UBound 0

  foreach cartera [

    set UBound UBound + item 0 [Utilidad] of proyectos with [id_proyecto = ? + 1]

  ]

  let j 0

  while [j < mayorhorizonte][

    let k 0
    let aux 0

    while [k < 4][

      set aux aux + item k item j matrizPrecios * read-from-string (item k item j matrizRecursos)

      set k (k + 1)
    ]

    set UBound UBound + aux

    set j (j + 1)
  ]

  set UBound precision UBound 5

  report UBound

end


to-report lowerBound [cartera2]

  let LBound 0
  let aux 0
  let aux2 0
  let aux3 0

  foreach cartera2 [

    set aux item 0 [totalAct] of proyectos with [id_proyecto = ? + 1]

    set aux2 item 0 [instanteFin] of sellos with [id_proyecto = ? and id_act = aux - 2]

    set aux3 item 0 item (aux - 1) item 0 [fechasLÃ-mite] of proyectos with [id_proyecto = ? + 1]
    ;; Duración del camino crítico

    set LBound LBound + (item 0 [NPVmax] of proyectos  with [id_proyecto = ? + 1]) -
                        (aux2 - aux3) * 0.05 * item 0 [NPVmax] of proyectos with [id_proyecto = ? + 1]

    ;; MÉTODO 2:
    ;; set LBound LBound + (item 0 [NPVmax] of proyectos with [id_proyecto = ? + 1]) /
                       ;;((1 + Tasa) ^ (aux2 - aux3)) ;; Tasa: variable global en pantalla del programa

   ]

  set LBound precision LBound 5

  report LBound


end
```

10. MÉTODO SERIE

```
to metodoSerie [cartera]

  let serie []
  let aux []

  foreach cartera [

    let vector sort-by < [who] of sellos with [_inicio != " " and id_proyecto = ?]

    ;; Sólo dimos valor a la propiedad _inicio al primer sello de cada actividad,para tener ahora un único
    ;; representante

    foreach vector [

      set aux []

      let aux1 [id_proyecto] of sello ? + 1
      let aux2 [id_act] of sello ?
      let aux3 [prioridad] of sello ?

      set aux lput aux1 aux
      set aux lput aux2 aux
      set aux lput aux3 aux

      set serie lput aux serie
    ]

  ] ;; Así represento todas las actividades por 3 í-ndices:
    ;; [proyecto / número de actividad / prioridad]



  ;; Ahora generamos varios criterios además del valor de la prioridad para ir intentando
  ;; introducir las actividades según el método Serie:

  let j 0
  let i 0

  while [j < 2][

  ;; Que si una actividad usa menos recursos, a pesar de tener peor valor de prioridad, puede preceder.
  ;; Realizamos 2 iteraciones sobre el vector "serie" original

    set i 0

    while [i < length serie - 1][

      if ( item 0 item 0 [recurso] of sellos with [id_proyecto = (item 0 item i serie - 1)
           and id_act = item 1 item i serie and prioridad = item 2 item i serie] <
           item 0 item 0 [recurso] of sellos with [id_proyecto = (item 0 item (i + 1) serie - 1)
           and id_act = item 1 item (i + 1) serie and prioridad = item 2 item (i + 1) serie])[

        let move item (i + 1) serie
        set serie remove item (i + 1) serie  serie
        set serie lput move serie

      ]

      set i (i + 1)
    ]

  set j (j + 1)
  ]


  set serie sort-by [item 2 ?1 < item 2 ?2] serie

  ;; Ordenar de menor a mayor valor de prioridad de la actividad. Según la expresión,
  ;; menor prioridad supone que la actividad requiere programarse antes


  if (Comentarios = true) [print (word "Ordenación actividades según prioridad " serie)]


  let h 0
```

```
while [h < 3][ ;; Valorar que un proyecto esté antes de finalizar e incluir sus actividades cuanto antes

  let testigo []

  set i 0

  while [i < item 0 [N_enCARTERA] of proyectos with [id_proyecto = 1]][
  ;; Vector testigo con tantos 0s como proyectos tiene la cartera

    set testigo lput 0 testigo

    set i (i + 1)

  ]

  set i 0

  while [i < length serie - 1][

    set j 0

    while [ j < length cartera ][
    ;; Conforme paso por las actividades de los proyectos, lo voy registrando.

      if (item 0 item i serie = j + 1) [set testigo replace-item j testigo (item j testigo + 1)]

      set j (j + 1)
    ]


    if (item (item 0 item i serie - 1) testigo < item (item 0 item (i + 1) serie - 1) testigo ) [

      ;; Sólo si una actividad tiene más actividades previas de ese proyecto ya colocadas,
      ;; la adelanto en mi lista de preferencias "serie".

      let move item i serie
      let move2 item (i + 1) serie

      set serie replace-item i serie move2
      set serie replace-item (i + 1) serie move

    ]

    set i (i + 1)
  ]

  set h (h + 1)
]

if (Comentarios = true) [print (word "Ordenación actividades permitiendo que proyectos terminen
                         cuanto antes "  serie)]


;; ____________________________________________________________________________
;; Representación gráfica y aplicación del método Serie ____________________
;; ____________________________________________________________________________


set i 0
let y 10

ask patches [set pcolor 0 set plabel ""]
ask patches with [pycor = 9 and pxcor >= 10 and pxcor < 10 + mayorhorizonte][set plabel (pxcor - 9)]

let m 1

while [m <= read-from-string (item 0 item 0 matrizRecursos)][

  ask patches with [pxcor = 9 and pycor = y + m - 1][ set plabel m ]

  set m (m + 1)

]


let finalizados []

while [i < (length serie) ][
```

```
  if (item 0 [In/Out] of sellos with [id_proyecto = (item 0 (item i serie) - 1) and
                                      id_act = item 1 (item i serie)] = "Out")[
    ;; La actividad está habilitada para ser programada

    let x cota_x (item i serie) ;; A partir de qué cota X puede ser programada

    let auxDur (item 0 [dura] of sellos with [id_proyecto  = (item 0 (item i serie) - 1) and
                  id_act = (item 1 (item i serie)) and prioridad = (item 2 (item i serie))])
    let auxRec (item 0 item 0 [recurso] of sellos with [id_proyecto  = (item 0(item i serie) - 1)
                and id_act = (item 1 (item i serie)) and prioridad = (item 2 (item i serie))])


    if (x + auxDur <= 10 + mayorhorizonte)[

      let w x

      while [ w < x + auxDur][ ;; Desde esa X hasta su duración, programa la actividad.

        set y cota_y w (item i serie) ;; A partir de qué cota Y podemos programarla

        let z y

        while [ z < y + auxRec ][

          if ([pcolor] of patches with [pxcor = w and pycor = z] = [0])[

            ask patches with [pxcor = w and pycor = z] [

              set pcolor item 0 [color] of proyectos with [id_proyecto = (item 0 (item i serie))]
              set plabel item 1 (item i serie)

            ]

          ]

          set z ( z + 1)
        ]

        set w ( w + 1)
      ]
    ]

  ]


  set i (i + 1)

]


  foreach cartera [

    if (all? sellos with [id_proyecto = ?] [In/Out = "In"])[

      print (word " [[ Concluye proyecto " (? + 1) " ]]")

      set finalizados lput ? finalizados

    ]

  ]

  set carteraMS []
  set carteraMS finalizados

  ask sellos [set In/Out " "]

  set LB lowerBound finalizados

  ;; De los proyectos que escogimos por tener utilidad positiva nos quedamos con
  ;; los que han conclui-do tras aplicar el método Serie


  actualizarPrecios ;; Respecto a la demanda global que hicieron los proyectos iniciales
                    ;; que tuvieron utilidad positiva en su programación individual

  guardaSolucion finalizados ;; Si es la mejor solución encontrada hasta el momento, la almacena

end
```

```
to-report cota_x [actividad]


  let antec (item 0 [actividadesAntecesoras] of proyectos with [id_proyecto = (item 0 actividad)])
  set antec (item (item 1 actividad - 1) antec)

  let restriccion [0]

  foreach antec [

    if(? != 0) [set restriccion lput (item 0 [instanteFin] of sellos with
               [id_proyecto = (item 0 actividad - 1) and id_act = (? - 1)]) restriccion ]

    ]

  if (restriccion != [])[set restriccion max restriccion]
  ;; No se puede programar antes que sus antecesoras, por eso la cota X empieza de "restricción" en adelante

  let testigo [0]
  let auxRec (item 0 item 0 [recurso] of sellos with [id_proyecto  = (item 0 actividad - 1)
                             and id_act = (item 1 actividad) and prioridad = (item 2 actividad)])
  let auxDur (item 0 [dura] of sellos with [id_proyecto  = (item 0 actividad - 1)
                             and id_act = (item 1 actividad) and prioridad = (item 2 actividad)])


  while [ item 0 testigo != 2][

    set testigo []
    let j 0
    let auxTime []

    while [j < auxDur][

      set auxTime lput (10 + restriccion + j) auxTime

      set j (j + 1)
    ]


    let cotasY []

    foreach auxTime [

      if( cota_y ? actividad != "No") [set cotasY lput ((cota_y ? actividad - 10) + auxRec -
                                       read-from-string (item 0 item 0 matrizRecursos)) cotasY]
      if( cota_y ? actividad = "No") [set cotasY lput "No" cotasY]

    ]


    foreach cotasY [

      ifelse (is-number? ?) [if (? > 0)[set testigo lput 1 testigo]] [set testigo lput 1 testigo]

    ]

    foreach cotasY [

      if (is-number? ?) [if (? <= 0)[set testigo lput 2 testigo]]

    ]

    set testigo sort-by < testigo

    if (item 0 testigo = 1)[set restriccion (restriccion + 1)]

  ]

  let x 10 + restriccion
  let y 10



  let w x

  while [w < 10 + mayorhorizonte and (item 0 [In/Out] of sellos with [id_proyecto = (item 0 actividad - 1) and
                                                                      id_act = (item 1 actividad)] != "In")][

    if (item 0 [In/Out] of sellos with [id_proyecto = (item 0 actividad - 1) and
                                        id_act = (item 1 actividad)] != "In")[
```

```
      if ( w + auxDur <= (10 + mayorhorizonte)) and (item 0 [In/Out] of sellos with
                            [id_proyecto = (item 0 actividad - 1) and id_act = (item 1 actividad)] != "In")[

        if (count patches with [pxcor = w and pycor >= y and pycor < y +
                                  read-from-string (item 0 item 0 matrizRecursos) and pcolor = 0] >= auxRec)[

          if (count patches with [pxcor >= w and pxcor < w + auxDur and pycor >= y and pycor < y +
                       read-from-string (item 0 item 0 matrizRecursos) and pcolor = 0] >= (auxDur * auxRec))[

            ask sellos with [id_proyecto = (item 0 actividad - 1) and id_act = (item 1 actividad)][

              set In/Out "In"
              set instanteFin (w + auxDur) - 10

            ]

            if ( (item 1 actividad) != (item 0 [totalAct] of proyectos with
                                                                [id_proyecto = item 0 actividad ] - 2))[

              let suces item (item 1 actividad) item 0 [actividadesSucesoras] of proyectos with
                            [id_proyecto = (item 0 actividad)]

              foreach suces [

                if (? != 0)[
                  set antec item (? - 2)(item 0 [actividadesAntecesoras] of proyectos with
                            [id_proyecto = (item 0 actividad)])
                  set testigo []

                  foreach antec [

                    if (? != 0) [ifelse (item 0 [In/Out] of sellos with
                                  [id_proyecto = (item 0 actividad - 1) and id_act = (? - 1)] = "In")
                                  [set testigo lput 1 testigo][set testigo lput 0 testigo]]
                    if (? = 0) [set testigo lput 1 testigo]
                  ]

                  set testigo sort-by < testigo

                  if ( item 0 testigo != 0)[

                    ask sellos with [id_proyecto = (item 0 actividad - 1) and
                                id_act = (? - 1)] [set In/Out "Out"]

                  ];; Habilitamos a las sucesoras de la actividad que acabamos de programar

                ]
              ]
            ]
          ]

        ]
      ]

    ]


    set w (w + 1)

  ]

  report (x)

end


to-report cota_y [x actividad]

 let y 10
 let auxRec (item 0 item 0 [recurso] of sellos with [id_proyecto  = (item 0 actividad - 1) and
             id_act = (item 1 actividad) and prioridad = (item 2 actividad)])

 let z y

  while [z < y + read-from-string (item 0 item 0 matrizRecursos)][

    if ([pcolor] of patches with [pxcor = x and pycor = z] = [0])[

      report z
```

```
      ]

    set z (z + 1)
  ]

  report "No"

end
```

11. ACTUALIZADOR PRECIOS

```
to actualizarPrecios ;; Paso variable

  let t 0
  let paso2 (paso / 2)
  let pasoV 0

  while [t < mayorhorizonte][

    let k 0
    let aux2 []

    while [k < 4][

      ifelse (item k (item t matrizDemanda) - read-from-string item k (item t matrizRecursos) > 0 )
             [set pasoV paso][set pasoV paso2]

      let aux1 item k (item t matrizPrecios) + pasoV *
               (item k (item t matrizDemanda) - read-from-string item k (item t matrizRecursos))

      ;; Los precios se actualizan en funciÃ³n de la demanda de recursos de la cartera tras ser escogidos
      ;; los proyectos con utilidad positiva, con respecto a los recursos globales que disponemos


      if (aux1 < 0 ) [set aux1 0]

      set aux2 lput aux1 aux2

      set k (k + 1)

    ]

    set matrizPrecios replace-item t matrizPrecios aux2

   set t (t + 1)
 ]

  stop

end


;to actualizarPrecios ;; Paso fijo
;
;  let t 0
;
;  while [t < mayorhorizonte][
;
;    let k 0
;    let aux2 []
;
;    while [k < 4][
;
;      let aux1 item k (item t matrizPrecios) + paso *
;               (item k (item t matrizDemanda) - read-from-string item k (item t matrizRecursos))
;
;
;      if (aux1 < 0 ) [set aux1 0]
;
;      set aux2 lput aux1 aux2
;
;      set k (k + 1)
;
;    ]
;
;    set matrizPrecios replace-item t matrizPrecios aux2
;
;   set t (t + 1)
; ]
;
;  stop
;
;end
```

```
to iterar

  let indicesProyecto sort-by < [id_proyecto] of proyectos
  set indicesProyecto map [? - 1] indicesProyecto

  set check 1

  while [ check < 150 and (length mejoraSol) < 40 and carteraInicial != [] ][

    update-plots

    print (word " UB: " UB)
    print (word " Mejor LB: " LB)

   ;; if ( user-yes-or-no? "Siguiente iteraciÃ³n")[

      print (word ":::::::::::::::::::: ITERACIÓN " (check + 2) " ::::::::::::::::::::")

      ask patches [set pcolor 0 set plabel " "]
      ask sellos [die]
      ask proyectos [set hidden? true]


      foreach indicesProyecto [

        iniciarGrafo ?

        PushRelabel ?

        iniciarGrafo ?

        CorteMinimo ?

      ]

      set check (check + 1)

    ;;]
  ]


  show date-and-time

  file-open "Experimentos.csv"

  file-print (word proy1 ", " proy2 ", " proy3 ", " proy4 ", " mayorhorizonte ", " item 0 item 0
matrizRecursos
              ", " paso ", " fin1 ", " fin2 ", " fin3 ", " fin4 ", " LB ", " iteracion ", " date-and-time)

  file-close

  repeat 2 [beep wait 1] ;; Sonido cuando finaliza


end


to guardaSolucion [finalizados]

  mejoraSolucion

  if ( LB > diferencia )[

    set mejoraSol []

    set diferencia LB

    set iteracion (check + 1)

    set solucion []

    foreach finalizados [

      set solucion lput (? + 1) solucion

    ]
```

```
;;export-interface (word (iteracion + 1) "Solución.png")

if (proy1 = 0)[set fin1 0]
if (proy2 = 0)[set fin2 0]
if (proy3 = 0)[set fin3 0]
if (proy4 = 0)[set fin4 0]


if (numProj = 2)[

  if (proy1 != 0) and (proy2 != 0)[

    set fin1 item 0 [instanteFin] of sellos with [id_proyecto = 0 and
                     id_act = (item 0[totalAct] of proyectos with [id_proyecto = 1] - 2)]
    set fin2 item 0 [instanteFin] of sellos with [id_proyecto = 1 and
                     id_act = (item 0[totalAct] of proyectos with [id_proyecto = 2] - 2)]

  ]

  if (proy1 != 0) and (proy3 != 0)[

    set fin1 item 0 [instanteFin] of sellos with [id_proyecto = 0 and
                     id_act = (item 0[totalAct] of proyectos with [id_proyecto = 1] - 2)]
    set fin3 item 0 [instanteFin] of sellos with [id_proyecto = 1 and
                     id_act = (item 0[totalAct] of proyectos with [id_proyecto = 2] - 2)]

  ]

  if (proy1 != 0) and (proy4 != 0)[

    set fin1 item 0 [instanteFin] of sellos with [id_proyecto = 0 and
                     id_act = (item 0[totalAct] of proyectos with [id_proyecto = 1] - 2)]
    set fin4 item 0 [instanteFin] of sellos with [id_proyecto = 1 and
                     id_act = (item 0[totalAct] of proyectos with [id_proyecto = 2] - 2)]

  ]

  if (proy2 != 0) and (proy3 != 0)[

    set fin2 item 0 [instanteFin] of sellos with [id_proyecto = 0 and
                     id_act = (item 0[totalAct] of proyectos with [id_proyecto = 1] - 2)]
    set fin3 item 0 [instanteFin] of sellos with [id_proyecto = 1 and
                     id_act = (item 0[totalAct] of proyectos with [id_proyecto = 2] - 2)]

  ]

  if (proy2 != 0) and (proy4 != 0)[

    set fin2 item 0 [instanteFin] of sellos with [id_proyecto = 0 and
                     id_act = (item 0[totalAct] of proyectos with [id_proyecto = 1] - 2)]
    set fin4 item 0 [instanteFin] of sellos with [id_proyecto = 1
                     and id_act = (item 0[totalAct] of proyectos with [id_proyecto = 2] - 2)]

  ]

  if (proy3 != 0) and (proy4 != 0)[

    set fin3 item 0 [instanteFin] of sellos with [id_proyecto = 0 and
                     id_act = (item 0[totalAct] of proyectos with [id_proyecto = 1] - 2)]
    set fin4 item 0 [instanteFin] of sellos with [id_proyecto = 1 and
                     id_act = (item 0[totalAct] of proyectos with [id_proyecto = 2] - 2)]

  ]
]

if (numProj = 3)[

  if (proy1 != 0) and (proy2 != 0) and (proy3 != 0)[

    set fin1 item 0 [instanteFin] of sellos with [id_proyecto = 0 and
                     id_act = (item 0[totalAct] of proyectos with [id_proyecto = 1] - 2)]
    set fin2 item 0 [instanteFin] of sellos with [id_proyecto = 1 and
                     id_act = (item 0[totalAct] of proyectos with [id_proyecto = 2] - 2)]
    set fin3 item 0 [instanteFin] of sellos with [id_proyecto = 2 and
                     id_act = (item 0[totalAct] of proyectos with [id_proyecto = 3] - 2)]

  ]

  if (proy1 != 0) and (proy2 != 0) and (proy4 != 0)[
```

```
      set fin1 item 0 [instanteFin] of sellos with [id_proyecto = 0 and
                       id_act = (item 0[totalAct] of proyectos with [id_proyecto = 1] - 2)]
      set fin2 item 0 [instanteFin] of sellos with [id_proyecto = 1 and
                       id_act = (item 0[totalAct] of proyectos with [id_proyecto = 2] - 2)]
      set fin4 item 0 [instanteFin] of sellos with [id_proyecto = 2 and
                       id_act = (item 0[totalAct] of proyectos with [id_proyecto = 3] - 2)]

    ]

    if (proy1 != 0) and (proy3 != 0) and (proy4 != 0)[

      set fin1 item 0 [instanteFin] of sellos with [id_proyecto = 0 and
                       id_act = (item 0[totalAct] of proyectos with [id_proyecto = 1] - 2)]
      set fin3 item 0 [instanteFin] of sellos with [id_proyecto = 1 and
                       id_act = (item 0[totalAct] of proyectos with [id_proyecto = 2] - 2)]
      set fin4 item 0 [instanteFin] of sellos with [id_proyecto = 2 and
                       id_act = (item 0[totalAct] of proyectos with [id_proyecto = 3] - 2)]

    ]

    if (proy2 != 0) and (proy3 != 0) and (proy4 != 0)[

      set fin2 item 0 [instanteFin] of sellos with [id_proyecto = 0 and
                       id_act = (item 0[totalAct] of proyectos with [id_proyecto = 1] - 2)]
      set fin3 item 0 [instanteFin] of sellos with [id_proyecto = 1 and
                       id_act = (item 0[totalAct] of proyectos with [id_proyecto = 2] - 2)]
      set fin4 item 0 [instanteFin] of sellos with [id_proyecto = 2 and
                       id_act = (item 0[totalAct] of proyectos with [id_proyecto = 3] - 2)]

    ]
  ]

  if (numProj = 4)[

    set fin1 item 0 [instanteFin] of sellos with [id_proyecto = 0 and
                     id_act = (item 0[totalAct] of proyectos with [id_proyecto = 1] - 2)]
    set fin2 item 0 [instanteFin] of sellos with [id_proyecto = 1 and
                     id_act = (item 0[totalAct] of proyectos with [id_proyecto = 2] - 2)]
    set fin3 item 0 [instanteFin] of sellos with [id_proyecto = 2
                     and id_act = (item 0[totalAct] of proyectos with [id_proyecto = 3] - 2)]
    set fin4 item 0 [instanteFin] of sellos with [id_proyecto = 3 and
                     id_act = (item 0[totalAct] of proyectos with [id_proyecto = 4] - 2)]


  ]

 ]

 stop

end


to mejoraSolucion

 set mejoraSol lput 1 mejoraSol

end
```

13. CORTES

```
to posiblesCortes [ID auxCorte ind_act j_anterior]

  let j j_anterior

  let aux12 0
  let aux123 0
  let aux1234 0

  let k 0

  while [k < length (item (ind_act - 1) item 0 [actividadesAntecesoras] of proyectos with [id_proyecto = ID +
1])][

    let aux1 item k item (ind_act - 1) item 0 [actividadesAntecesoras] of proyectos with [id_proyecto = ID +
1]
    set aux12 item (aux1 - 1) item 0 [duracionAct] of proyectos with [id_proyecto = ID + 1]
    set aux1 item (aux1 - 2) auxCorte + item (aux1 - 1) item 0 [duracionAct] of proyectos with [id_proyecto =
ID + 1]

    if (aux1 > aux123) [set aux123 aux1 set aux1234 aux12]
    if (aux1 = aux123 and aux12 > aux1234) [set aux1234 aux12]

    set k (k + 1)
  ]



  let aux4 0
  let aux5 0
  let aux6 0

  set j (j + aux1234)

  while [ j <= item 1 item (ind_act + 1) item 0 [fechasLímite] of proyectos with [id_proyecto = ID + 1] ][

    let aux lput j auxCorte

    if ( (ind_act + 1) < item 0 [totalAct] of proyectos with [id_proyecto = ID + 1] - 2 ) [

      set k 0

      while [k < length (item (ind_act) item 0 [actividadesAntecesoras] of proyectos with
            [id_proyecto = ID + 1])][

        let aux3 item k item (ind_act) item 0 [actividadesAntecesoras] of proyectos with
                 [id_proyecto = ID + 1]
        set aux4 item (aux3 - 2) aux
        set aux3 item (aux3 - 2) aux + item (aux3 - 1) item 0 [duracionAct] of proyectos with
                 [id_proyecto = ID + 1]

        if (aux3 > aux6) [ set aux6 aux3 set aux5 aux4]
        if (aux3 = aux6 and aux4 < aux5) [set aux5 aux4]

        set k (k + 1)
      ]

      posiblesCortes ID aux (ind_act + 1) aux5

    ]

    if (length aux = item 0 [totalAct] of proyectos with [id_proyecto = ID + 1] - 2)
       [set posiblesRutas lput aux posiblesRutas]

    set j (j + 1)
  ]

end
```

14. SETUP GRAFO

```
to iniciarGrafo [ID]

  ask nodos with [id_proyecto = ID][set color item 0 [color] of proyectos with [id_proyecto = ID + 1] ]
  ask links with [id_proyec = ID][ set flujo 0 set color 5 set admisible 0 set thickness 0]


  ;; Actualizar las capacidades de los arcos de asignación en función de los precios (como en el grafo
original).
  ;; Reset al grafo para un nuevo Push Relabel:

  let i 1
  let j 0

  while [i <= item 0 [totalAct] of proyectos with [id_proyecto = ID + 1] - 2][

    set j 0

    while [j <= item 0 [horiz] of proyectos with [id_proyecto = ID + 1]][

      if (j >= item 0 item i item 0 [fechasLímite] of proyectos with [id_proyecto = ID + 1] and
          j <= item 1 item i item 0 [fechasLí-mite] of proyectos with [id_proyecto = ID + 1] + 1)[

        ask links with [id_proyec = ID and id_act_ini = (i + 1) and id_act_fin = (i + 1) and _slot = j][

          let h j
          set capacidad 0
          set admisible 0

          while [h < j + item 0 [duracion] of nodos with [id_proyecto = ID and id_act = (i + 1) and
                 slot = j]][

            let k 0

            while [k < 4][

              set capacidad precision (capacidad + item k item h matrizPrecios *
                            item k item i item 0 [necesidadesAct] of proyectos with [id_proyecto = ID + 1])5
              set residual precision (capacidad - flujo) 5
              set label (word capacidad "    ")

              set k ( k + 1)
            ]

            set h (h + 1)
          ]

          if (i = item 0 [totalAct] of proyectos with [id_proyecto = ID + 1] - 2)[

            ;; Si se trata de la última actividad, añade la penalización en función de cuando terminará
            ;; el proyecto respecto a la fecha en la que podía haber terminado

            set capacidad precision (capacidad + 0.05 * item 0 [NPVMax] of proyectos with
                                     [id_proyecto = ID + 1] * (j - item 0 item i item 0 [fechasLímite]
                                      of  proyectos with [id_proyecto = ID + 1])) 5
            set residual precision (capacidad - flujo) 5
            set label (word capacidad "    ")
          ]

        ]

      ]

      set j (j + 1)
    ]
    set i (i + 1)
  ]

  ask nodos with [label = "S  " and id_proyecto = ID][set _label count nodos with [id_proyecto = ID]]

  ask nodos with [label = "T  " and id_proyecto = ID][set _label 0]

  ask nodos with [label != "S  " and label != "T  " and id_proyecto = ID]
                 [set _label 0 set exceso 0 set activo 0]


End
```