(19) OFICINA ESPAÑOLA DE PATENTES Y MARCAS

ESPAÑA

(11) Número de publicación: **2 726 806**

(51) Int. CI.:

G06F 9/48 (2006.01)

(12) TRADUCCIÓN DE PATENTE EUROPEA T3

(86) Fecha de presentación y número de la solicitud internacional: **13.12.2012 PCT/EP2012/075468**

(87) Fecha y número de publicación internacional: **04.07.2013 WO13098088**

(96) Fecha de presentación y número de la solicitud europea: **13.12.2012 E 12808341 (7)**

(97) Fecha y número de publicación de la concesión europea: **06.03.2019 EP 2798493**

(54) Título: **Programador de trabajos para sistema electromecánico de análisis biológicos**

(30) Prioridad:

30.12.2011 EP 11196275

(45) Fecha de publicación y mención en BOPI de la traducción de la patente:
09.10.2019

(73) Titular/es:

BIOMERIEUX (100.0%)
69280 MARCY L'ETOILE, FR

(72) Inventor/es:

VICARIO, ENRICO;
RIDI, LORENZO;
CARIGNANO, ANDREA y
TORRINI, JACOPO

(74) Agente/Representante:

ELZABURU, S.L.P

DESCRIPCIÓN

Programador de trabajos para sistema electromecánico de análisis biológicos

Campo de la tecnología

La presente invención se refiere a un método y sistema para programar una pluralidad de secuencias concurrentes de trabajos sincronizados que están sujetos a restricciones de exclusión mutua y retraso mutuo. En particular, la presente invención se refiere a un programador para un sistema que realiza múltiples análisis biológicos concurrentes que aplican diferentes protocolos de análisis en diferentes muestras mientras comparten un conjunto de dispositivos electromecánicos. El artículo de G.Bucci; L.Carnevali; L.Ridi y E.Vicario "Oris: a Tool for Modeling, Verification and Evaluation of Real-Time Systems", International Journal of software tools for technology transfer, vol.12, no.5, 2010, páginas 391-403, describe un método de programación de la técnica anterior.

Antecedentes

Si bien el problema de programar una pluralidad de secuencias de trabajo concurrentes y sincronizadas con restricciones de retraso mutuo y exclusión mutua con intervalos de tiempo limitado puede aplicarse a una gran variedad de contextos, en la presente solicitud, sin pérdida de generalidad, abordamos el problema con referencia a un ejemplo concreto de un programador de tiempo de ejecución para un sistema electromecánico para análisis biológicos, como por ejemplo el sistema VIDAS diseñado por bioMerieux.

El sistema del presente ejemplo es capaz de ejecutar un grupo de múltiples análisis concurrentes, cada uno alternando las etapas de la reacción biológica de la muestra y las operaciones de transferencia realizadas por dispositivos electromecánicos compartidos; la duración de cada etapa de reacción está determinada por el protocolo biológico del análisis, pero se pueden agregar tiempos de espera limitados después de completar cada etapa de reacción. Esto plantea el problema de la programación de determinar los tiempos de espera después de cada etapa para cumplir con los retrasos máximos permitidos y para evitar conflictos en el uso de los recursos compartidos, al tiempo que se maximiza el número de análisis concurrentes y se minimiza el tiempo total de finalización.

Con más detalle, cada tipo de análisis biológico está compuesto por una fase de pretratamiento y un protocolo analítico. Al comienzo del análisis, la muestra está contenida en un tubo de ensayo identificado de forma única por un código de barras, mientras que otros tubos contienen fluidos de dilución e incubación y otros tubos están vacíos. Durante la fase de pretratamiento, una pipeta automática vierte repetidamente la muestra entre varios tubos; Cada acción de la pipeta dura un tiempo determinista. Se permiten tiempos de espera entre acciones sucesivas, pero se limitan a variar entre los valores mínimos y máximos determinados por los períodos de incubación / reacción y las propiedades de deterioro de la muestra. Después de completar la fase de pretratamiento, el protocolo analítico sigue una secuencia fija de etapas durante los cuales la muestra se combina con reactivos y se realizan múltiples mediciones por un cabezal de lectura cuyas operaciones toman duraciones deterministas (véase la figura 1).

Para una explotación eficiente de los componentes electromecánicos, se llevan a cabo múltiples análisis, posiblemente de diferentes tipos, al mismo tiempo. Para este fin, el sistema compone múltiples secciones, una para cada análisis. Cada sección se divide a su vez en posiciones, que llevan diferentes muestras que pueden someterse a diferentes tratamientos previos y pueden operar en muestras de diferentes sujetos. Sin embargo, dado que el cabezal de lectura está diseñado para tomar medidas en una sección entera a la vez, todas las posiciones en la misma sección están restringidas para ejecutar el mismo protocolo analítico (véase la figura 2).

La pipeta y el cabezal de lectura se comparten entre diferentes posiciones y secciones y no pueden ser utilizados simultáneamente por dos análisis diferentes. Esto plantea un problema de programación que debe determinar los valores factibles del retraso inicial de cada tratamiento previo y los tiempos de espera entre los eventos del mismo análisis. Tales valores deben evitar la necesidad de acciones contemporáneas de la pipeta y del cabezal de lectura, evitar tiempos de espera más allá de los límites permitidos y mantener el tiempo total del grupo de análisis lo más corto posible.

Formulación del problema

El problema se puede plantear formalmente en la notación general de la programación del taller como un conjunto de n trabajos independientes y no preferibles J_1 .... J_n, con tiempos de liberación r_1 .... r_n y tiempos de ejecución deterministas e_1 .... e_n, cada trabajo se asigna estáticamente a una de cada m máquinas independientes M_1 .... M_n, sujeto a restricciones de precedencia y exclusión mutua:

- para cualquiera de los dos trabajos J_i y J_k, una restricción de precedencia puede requerir que el retraso entre la finalización de J_k y el comienzo de J_i rango dentro de un mínimo d^-_{ik} y un máximo d^+_{ik}:

$$d^-_{ik} \leq r_i - (r_k + e_k) \leq d^+_{ik}$$

- para cualquiera de los dos trabajos J_i y J_k, una restricción de exclusión mutua puede requerir que los períodos de ejecución de J_k y J_i no se superponen esto se puede lograr con cualquiera de las siguientes dos

determinaciones, que garantizan que J_i comienza después de la finalización de J_k o viceversa:

$$r_i \geq r_k + e_k \text{ o } r_i + e_i \leq r_k$$

El problema de la programación equivale a determinar los tiempos de liberación r_1 .... r_n, sujeto a todas las restricciones prescritas de precedencia y exclusión mutua, a fin de minimizar el tiempo de finalización de la pluralidad general de las secuencias de trabajo. La Figura 3a ilustra una instancia del problema: los Trabajos J_1, J_2, J_3 y J_4 se asignan en las máquinas M_1 y M_2; el Trabajo J_2 tiene una exclusión mutua con respecto a J_4 y una restricción de precedencia que limita su tiempo de inicio con respecto al tiempo de finalización de J_1 y J_3. Una restricción de precedencia también limita el retraso del tiempo de inicio de J_4 con respecto a la finalización de J_3. La figura 3b muestra una solución para el problema.

El problema de programación definido anteriormente se caracteriza por el entrelazamiento de tiempos de lanzamiento no deterministas con restricciones de exclusión y precedencia mutua, y por lo tanto puede formularse convenientemente sobre la base de formalismos no deterministas como las Redes de Petri Sincronizadas (TPN) o los Autómatas Sincronizados (TA) a través de una traducción directa de conceptos de programación como trabajos, restricciones y recursos, en lugares / ubicaciones, transiciones, condiciones previas y posteriores. Esto abre el camino a la aplicación de herramientas de modelado y verificación que administran la combinación de restricciones de concurrencia y sincronizadores no deterministas a través del análisis simbólico del espacio de estado basado en las Regiones (véase R. Alur, D.L. Dill: Automata for modeling real-time systems . En Proc. Of 17th ICALP, 1990) o las zonas de Matriz de Límite de Diferencia (DBM) (véase D. Dill. Timing Assumptions and Verification of Finite-State Concurrent Systems. En Proc. Workshop on Computer Aides Verification Methods for Finite State Systems, 1989). En esta perspectiva, el problema de la programación se puede formular como la identificación de un testigo en el espacio de estado simbólico que satisface una fórmula de Comprobación del Modelo en Tiempo Real (véase, por ejemplo, A. Fehnker. Scheduling a steel plant with timed automata. En Real-Time Computing Systems and Applications, 1999. RTCSA '99. Sexta Conferencia Internacional, páginas 280 -286, 1999) que selecciona "todos los comportamientos que alcanzan un estado de finalización dentro de un tiempo T máximo, después de atravesar solo estados sin conflictos en el uso de los recursos". El valor óptimo de T se puede obtener a través de la iteración polinomial, que se basa en varias herramientas y, en particular, en el entorno Uppaal (véase, por ejemplo, K. Larsen. Resource-efficient scheduling for real time systems. En Rajeev Alur e Insup Lee, editores, Embedded Software, volumen 2855 de Lecture Notes in Computer Science, páginas 16-19. Springer Berlin-Heidelberg, 2003.; o I. AlAttili, F. Houben, G. Igna, S. Michels, F. Zhu y F. W. Vaandrager. Adaptive scheduling of data paths using upaal tiga. En QFM, páginas 1-11, 2009). En la herramienta Oris (G. Bucci y L. Carnevali y L. Ridi y E. Vicario. Oris: a Tool for Modeling, Verification and Evaluation of Real-Time Systems . International Journal of Software Tools for Technology Transfer, 12 (5): 391 - 403, 2010), el valor óptimo T se deriva de manera directa evaluando el tiempo mínimo y máximo empleado en cada secuencia de transiciones que satisfacen las restricciones de secuencia esperadas (E. Vicario. Static Analysis and Dinamic Steering of Time Dependent Systems Using Time Petri Nets . IEEE Trans. en SW Eng., 27 (1): 728-748, agosto de 2001).

Desafortunadamente, la aplicación directa de este tipo de técnicas de análisis de espacio simbólico de estado hacen frente a los problemas de explosión de espacio de estado (a medida que el número de estados crece exponencialmente con el número de trabajos) y la maldición de la dimensionalidad (a medida que el tamaño de las estructuras de datos que codifican cada estado crece de forma cuadrática con el número de secuencias). Esto da como resultado una complejidad de memoria y tiempo que, de lejos, no es asequible para un programador en línea, incluso si el cálculo se realiza en un PC convencional, lo que a menudo no es el caso en aplicaciones industriales.

Solo para averiguar el orden de magnitud de la complejidad práctica, se experimentó el análisis del espacio de estado basado en zonas de DBM con la herramienta Oris en la representación de la TPN de un caso simplista compuesto por dos Secciones, tres Posiciones por Sección y retrasos deterministas: la enumeración de la TPN El procedimiento produciría un enorme espacio de estado de 255128 clases de estado de DBM, que requiere unos 20 minutos en un ordenador de escritorio de doble núcleo de 2 GHz, con 4216 trayectorias posibles desde el estado inicial hasta el nodo terminal final.

Incluso aplicando técnicas de vanguardia para reducir la complejidad a través de la simplificación del modelo, e incluso finalizando la partición del espacio de estado a través de Zonas de DBM (que son mucho más compactas que las Regiones), los tiempos de procesamiento y la demanda de memoria de esta clase de técnicas de análisis de espacio de estado no son compatibles con el uso previsto dentro de una aplicación industrial como el descrito anteriormente: esto requiere que el problema de programación se resuelva en línea en un enfoque casi en tiempo real, y que su ejecución sea asequible dentro de un componente integrado que se ejecuta en recursos de procesamiento y memoria limitados. Los recursos de hardware normalmente disponibles en este tipo de máquinas de pruebas biológicas (pero más generalmente en máquinas electromecánicas industriales) son muy limitados y no son compatibles con la gran demanda de una implementación de algoritmo matemático tan complejo.

Por lo tanto, sería altamente deseable una solución que pudiera cambiar la generalidad del análisis por eficiencia en el problema específico para ejecutarse más rápido y con menor demanda de memoria y potencia de procesamiento.

Objetos de la descripción

Es un objeto de la presente descripción superar al menos algunos de los problemas asociados con la técnica anterior.

Un objeto adicional de la presente invención es proporcionar un método que requiera memoria y tiempo de procesamiento limitados para calcular un programa aceptable y eficiente para una pluralidad de secuencias concurrentes de etapas sincronizadas sujetas a restricciones de exclusión mutua y retraso mutuo y que puedan retrasarse tiempos de espera acotados.

Esto se representa como un problema de programación que surge en un sistema electromecánico industrial real, que incluye encontrar sincronizaciones factibles para las acciones realizadas por un conjunto de componentes electromecánicos compartidos para completar un conjunto de análisis biológicos concurrentes en el mínimo tiempo posible y al mismo tiempo satisfacer la exclusión mutua y restricciones mutuas de la negación. En el ejemplo específico aquí descrito, se requiere que el algoritmo de programación se ejecute a bordo del hardware enviado con un sistema electromecánico actualmente equipado con una CPU de 200MHz y con menos de 10 MB de memoria. Además, para garantizar una capacidad de respuesta adecuada para los operadores que interactúan con el sistema, se permite que el algoritmo encuentre soluciones subóptimas pero eficientes, pero se requiere que finalice de manera concluyente dentro de un corto período de tiempo en el orden de unos pocos segundos.

Otro objeto de la invención es proporcionar un sistema electromecánico que ejecute una pluralidad de análisis biológicos concurrentes, cada uno de los periodos alternos de las operaciones de reacción y transferencia realizadas por dispositivos compartidos bajo las restricciones de sincronización y retraso mutuo determinadas por el protocolo de análisis biológico, y bajo las restricciones de exclusión mutua determinadas por el uso exclusivo de dispositivos compartidos.

Compendio

La presente divulgación proporciona un método de programación según lo establecido en las reivindicaciones adjuntas.

De acuerdo con una primera realización, la presente divulgación proporciona un método de programación, en un sistema adaptado para ejecutar una pluralidad de trabajos no preferibles J_1 .... J_n, con tiempos de ejecución deterministas e_1 .... e_n, cada trabajo que está asignado estáticamente a una de m máquinas independientes M_1 .... M_m, para determinar un tiempo de liberación r_1 .... r_n para cada trabajo J_1 .... J_n a fin de minimizar el tiempo de finalización del conjunto general de trabajos, respetando la siguiente restricción de precedencia:

- para un conjunto predeterminado de pares J_x y J_y de la pluralidad de puestos de trabajo, el retraso entre la finalización de J_y y el comienzo de J_x rango dentro de un retraso mínimo d^-_{xy} y un retraso máximo d^+_{xy}:

  $$d^-_{xy} \leq r_x - (r_y + e_y) \leq d+_{xy}$$

  y las siguientes restricciones de exclusión:

- para un conjunto predeterminado de pares J_i y J_k de la pluralidad de trabajos, se cumple una de las siguientes dos determinaciones de restricción de exclusión:

  $$r_i \geq r_k + e_k \text{ o } r_i + e_i \leq r_k$$

  de forma que los periodos de ejecución de J_k y J_i no se superpongan,

  el método que comprende las etapas de: A) crear un conjunto de fronteras que incluya una zona de Matriz de Límite de Diferencia (DBM) que recopila los tiempos de liberación que satisfacen las restricciones de precedencia; B) seleccionar una de las zonas de DBM y eliminándola del conjunto de fronteras: C) responder a la zona de DBM seleccionada y no satisfacer al menos una restricción de exclusión de un par J_i y J_k del conjunto predeterminado de pares, construyendo una zona restringida de la zona de DBM seleccionada por cada una de las dos determinaciones que resuelven el conflicto ($r_i \geq r_k + e_k$ o $r_i + e_i \leq r_k$); D) comprobar si las zonas de DBM restringidas no están vacías y agregar las zonas de DBM restringidas no vacías al conjunto de fronteras; E) repetir las etapas B a D hasta que una zona de DBM seleccionada satisfaga todas las restricciones de exclusión.

El método de la presente invención permite la programación de secuencias concurrentes de trabajos sincronizados con restricciones de exclusión mutua y retraso, capturando conjuntos de posibles soluciones equivalentes a través de las zonas de Matriz de Límite de Diferencia (DBM) y al organizar la identificación de una solución factible como una búsqueda en una estructura grafo-teórica. Esto permite una serie de optimizaciones algorítmicas y enfoques heurísticos para acelerar la búsqueda de soluciones mientras se mantiene una representación exacta (es decir, no cuantificada ni aproximada) del conjunto de posibles configuraciones de sincronización.

De acuerdo con otra realización, la etapa de repetición se realiza hasta que todas las zonas de DBM en el conjunto

de fronteras satisfacen todas las restricciones de exclusión. Alternativamente, la etapa de repetición se detiene cuando se alcanza un tiempo de búsqueda máximo predeterminado. En otra realización, se selecciona una solución óptima (programa) dentro de todas las zonas de DBM que satisfacen todas las restricciones de exclusión.

Según otra realización, la selección de la etapa B se basa en una evaluación heurística, incluida la denominada heurística de intervalo más largo (segura) o la denominada heurística de intervalo más corto (codiciosa).

De acuerdo con otra realización, se proporciona un programa informático para realizar el método.

Según una realización adicional de la invención, se proporciona un sistema que implementa el método anterior. Dicho sistema se puede usar, por ejemplo, en un aparato electromecánico para realizar análisis biológicos.

La presente invención ofrece una serie de beneficios. Una de las ventajas de una realización preferida de la presente invención es que una operación de programación compleja se puede realizar con un tiempo y recursos de hardware limitados. En principio, el problema puede formularse como un caso de Comprobación de Modelos en Tiempo Real y resolverse a través de enfoques de análisis de espacio de estado bien establecidos. Sin embargo, tales enfoques enumerativos no son viables en la configuración específica, debido a las restricciones de hardware impuestas por la plataforma integrada esperada. Con el método de la presente invención, una adaptación a propósito directa de la teoría de DBM reduce en gran medida la complejidad del algoritmo, también al permitir la adopción de heurísticas que conducen la búsqueda hacia soluciones factibles. En una realización preferida de la presente invención, los refinamientos algorítmicos posteriores permiten reducir aún más los tiempos de cálculo, alcanzando los requisitos de rendimiento esperados.

Una ventaja adicional del método y el sistema de acuerdo con una realización de la presente invención es que permite el caso en el que uno o más análisis se agregan a un programa en curso al reclamar los recursos no utilizados sin poner en peligro la ejecución de los análisis ya programados.

Breve descripción de los dibujos

Ahora se hará referencia, a modo de ejemplo, a los dibujos adjuntos, en los que:

La Figura 1 es una representación esquemática de la arquitectura física de los componentes del sistema que ejecutan un solo análisis en un aparato electromecánico para realizar pruebas biológicas;

La Figura 2 es una representación esquemática de la arquitectura física del sistema general que compone múltiples análisis en posiciones y secciones en un aparato electromecánico para realizar pruebas biológicas de acuerdo con una realización de la presente invención;

La Figura 3a es una representación esquemática de un caso del problema de planificación para determinar los tiempos de liberación r_1 .... r_n, sujeto a todas las restricciones prescritas de precedencia y exclusión mutua, a fin de minimizar el tiempo de finalización del conjunto general de trabajos. La Figura 3b es una posible solución que determina el valor de los tiempos de liberación;

La Figura 4a muestra una DBM bidimensional que codifica el conjunto de soluciones para un par de variables t_i y t_j. La Figura 4b muestra una DBM bidimensional que destaca la presencia de una restricción no efectiva. La Figura 4c muestra la representación teórica gráfica de una DBM que involucra las tres variables t_i, t_j y T_k.

La Figura 5 muestra un ejemplo del árbol binario que está constituido por elecciones subsiguientes para las restricciones de exclusión del método de planificación de acuerdo con una realización preferida de la presente invención;

La Figura 6 muestra los resultados de experimentos de pruebas de rendimiento con dos heurísticas diferentes, de acuerdo con una realización preferida de la presente invención.

Descripción detallada de realizaciones preferidas

Para el propósito del tratamiento posterior, presentamos aquí los conceptos principales en los que se basa la solución de programación: Matrices de Límite de Diferencia (DBM) y algoritmo de Floyd-Warshall.

Una DBM codifica un sistema de desigualdades que identifica un conjunto de soluciones para un vector de N variables $T = \langle T_1 \ldots T_N \rangle$:

$$D = \begin{cases} \tau_i - \tau_j \leq b_{ij} & i, j \in [1 \ldots N] \cup *, b_{ij} \in \mathbb{Q} \\ \tau_* = 0 \end{cases}$$

Cualquier proyección bidimensional de una DBM da como resultado un poliedro del tipo representado en la Fig. 4a. Una DBM puede no estar completamente formada cuando una o más restricciones no son efectivas (véase la figura 4b); en este caso, un número infinito de valores para estas restricciones produce el mismo conjunto de soluciones. El

valor límite, de modo que cualquier restricción adicional cambiaría el conjunto de soluciones, se ha comprobado que existe y es único si y solo si se cumple la siguiente desigualdad triangular:

$$b_{ij} \leq b_{ik} + b_{kj} \; \forall \; i,j,k$$

Esta condición identifica el concepto de *forma normal*, que se define como la representación que satisface la desigualdad triangular.

El conjunto de restricciones de una DBM se puede convertir convenientemente en una formulación gráfico-teórica (véase la figura 4c) en la que los nodos del gráfico representan variables, y los bordes se etiquetan con valores de coeficientes de DBM (es decir, el borde que conecta los nodos correspondientes a Las variables i y j están etiquetadas con coeficiente b_{ij}): en esta perspectiva, la forma normal de un DBM se convierte en un problema de ruta más corta donde b_{ij} los coeficientes se calculan como la trayectoria más corta desde el nodo *i* a *j*.

Se comprueba fácilmente que la suposición de que DBM no está vacío excluye la existencia de *ciclos negativos* (es decir, ciclos en la gráfica cuyos coeficientes suman un valor negativo). Así, el problema puede ser resuelto por *N* Aplicaciones repetidas del conocido algoritmo de Dijkstra, con complejidad total O (N. N^2). El algoritmo Floyd-Warshall proporciona una solución directa que aún se ejecuta en $O(N^3)$, pero realiza operaciones mucho más simples que Dijkstra dentro de su ciclo interno y es preferible cuando se trata de matrices de adyacencia densa, como las correspondientes a los dominios DBM (véase E. Vicario. Static Analysis and Dynamic Steering of Time Dependent Systems Using Time Petri Nets. IEEE Trans. en SW Eng., 27 (1): 728-748, 2001).

Informamos en Alg. 1 el pseudocódigo de Floyd Warshall. Su núcleo invariante es variable.

b_{ij}^{k}, que codifica "la trayectoria más corta de j a i bajo la restricción de que solo los vértices con un índice mayor que k se pueden visitar como nodos intermedios". Bajo esta declaración, está claro que se puede dar a b_{ij}^{N} los valores iniciales de los coeficientes b_{ij}^{N}, lo que da sentido al primer bucle for del código. También está claro que b_{ij}^{0} comprende la solución final del problema, que en el código se logra al disminuir k de N a 1 a través del segundo bucle for. El núcleo real del algoritmo está, por lo tanto, restringido a la forma en que el bucle doble anidado en *i, j* es capaz de derivar b_{ij}^{k-1} desde b_{ij}^{k}.

ALGORITMO 1: ALGORITMO DE FLOYD-WARSHALL

```
1  begin
2      for [i, jj ∈ [0, N − 1] × [0, N − 1] do
3          b_ij^{N−1} = b_ij;
4      end
5      for k = N − 1 → 0 do
6          for [i, j] ∈ [0, N − 1] × [0, N − 1] do
7              b_ij^{k−1} = min{b_ij^k, b_ik^k + b_kj^k};
8          end
9      end
10 end
```

Las estructuras de datos de la Matriz de Límite de Diferencia y el algoritmo Floyd-Warshall constituyen el terreno básico para las herramientas de Verificación de propósito general y de Comprobación de Modelo, que en principio podrían proporcionar la funcionalidad para resolver el problema de programación. En una realización preferida de la presente invención, con el fin de restringir la complejidad del análisis del espacio de estado de un modelo general, adaptamos el núcleo algorítmico de las Zonas de la Matriz de Límite de Diferencia (DBM) a la identificación de los tiempos que resultan de diferentes ordenamientos entre las acciones sujetas a restricciones de exclusión mutua aprovechando la estructura regular específica de las secuencias programadas. Esto dio lugar a varias adaptaciones optimizadas del algoritmo Floyd-Warshall que se utilizaron para mantener las Zonas DBM en forma normal.

Luego organizamos DBM correspondientes a diferentes determinaciones de diferentes conjuntos de restricciones de exclusión mutua en una estructura gráfico-teórica que permite un recorrido incremental. En términos generales, el algoritmo construye de forma iterativa soluciones candidatas agregando restricciones de exclusión y verificando la conformidad de los parámetros temporales con las restricciones impuestas por el problema; en esta configuración, las estructuras de DBM se utilizan para codificar el conjunto de restricciones que determinan las precedencias y exclusiones entre los trabajos, mientras que el algoritmo Floyd-Warshall se invoca repetidamente para mantener las DBM en su forma normal, a fin de permitir la manipulación y comparación (eficiente) de resultados obtenidos.

El algoritmo inicialmente construye una zona DBM D_{prec} que representa todas las restricciones de precedencia del problema. Luego, restringe repetidamente la zona D_{prec} añadiendo cada restricción de exclusión en sus dos determinaciones diferentes, y descartando las zonas restringidas que resultan estar vacías. Cualquier zona no vacía

que se haya restringido con cierta determinación para todas las restricciones de exclusión comprende un conjunto de soluciones para el problema de programación. Esta solución corresponde de manera unívoca a una única determinación de restricciones de exclusión, y su zona DBM contiene todos y solo las temporizaciones que son consistentes con estas determinaciones. La sincronización óptima puede derivarse de manera directa y directa de la forma normal de la zona DBM.

En esta perspectiva, nuestro algoritmo de programación puede considerarse como el recorrido de una estructura gráfico-teórica, donde cada nodo de la estructura está asociado con una Zona DBM. En particular: la raíz es la zona D_{prec}, que satisface todas las precedencias pero no exclusiones; nodos intermedios restringen D_{prec} con un subconjunto de restricciones de exclusión en alguna determinación; nodos de la hoja restringen D_{prec} con todas las restricciones de exclusión en alguna determinación; cada borde de la estructura gráfico-teórica representa la elección para cierta determinación de una o más restricciones de exclusión. Sin pérdida de generalidad, el Alg. 2 informa el pseudocódigo de algoritmo de una implementación de acuerdo con una realización preferida de la presente invención: en el caso específico, cada borde agrega una única restricción de exclusión mutua, de modo que la estructura gráfico-teórica se convierte en un árbol binario.

ALGORITMO 2: PROGRAMANDO ALGORITMO

```
1 begin
2  Dprec = DBM encoding all precedence constraints;
3  Normalize(Dprec);
4  Enqueue(<Dprec, 0>);
5  Dsol = dummy solution;
6  while QueueIsNotEmpty() and time limit is not exceeded do
7      <D, c> = Dequeue();
8      Dr = AddRightConstraint(D, c + 1);

9      Normalize(Dr);
10     if (IsNotEmpty(Dr)) then
11         if (c + 1 == C) then
12             Dsol = GetBestSolution(Dr, Dsol);
13         else
14             Enqueue(<Dr, c + 1>);
15     Dl = AddLeftConstraint(D, c + 1);
16     Normalize(Dl);
17     if (IsNotEmpty(Di)) then
18         if (c + 1 == C) then
19             Dsol = GetBestSolution(Dl , Dsol);
20         else
21             Enqueue(<Dl, c + 1>);
22     end
23 end
24 return Dsol ;
25 end
```

La implementación se basa en algunas estructuras de datos e invariantes:

- El índice *c* representa el número de restricciones de exclusión agregadas a la restricción, que también toma el significado de profundidad en el árbol binario; además, *C* es el número de todas las restricciones de exclusión, de modo que un nodo en la profundidad *C* es una hoja.

- D_{sol} representa la mejor solución identificada en cualquier punto del cómputo.

  Se inicializa con una valoración ficticia, que podría ser, por ejemplo, un programa trivial e ineficiente o algún valor simbólico que pueda considerarse peor que cualquier otro.

- El algoritmo se basa en una cola que contiene nodos en el árbol, cada uno identificado por una zona *D* de DBM y un índice de profundidad *c*.

- Se supone que las restricciones de exclusión se ordenan de modo que un índice *c* identifique de forma única una restricción de exclusión.

El algoritmo también se basa en varias funciones:

- Normalizar (*D*) aplica el algoritmo Floyd-Warshall para normalizar el DBM *D* tomado como parámetro;

- Enqueue (<D, c>) y Dequeue () realizan respectivamente las operaciones de adición y extracción en la cola;

QueueIsNotEmpty () comprueba si la cola contiene al menos un elemento;

- AddRightConstraint (*D, c*) restringe *re* agregando la restricción del orden c en su determinación correcta (es decir, si la restricción está entre los trabajos Ji y Jj, entonces la restricción agregada es $ri - rj \geq ej$); de la misma manera, AddLeftConstraint (*D, c*) restringe *D* agregando la restricción del orden *c* en su determinación izquierda (es decir, si la restricción está entre los trabajos Ji y Jj, entonces la restricción agregada es $r_j - r_i \geq e_i$).

- GetBestSolution (*D', D"*) recibe dos zonas DBM y selecciona la que incluye la mejor sincronización, por ejemplo, la sincronización que minimiza el tiempo de finalización general para todas las secuencias de trabajos.

El ejemplo descrito anteriormente se basa en la elección entre un par de determinaciones para cada restricción de exclusión mutua para cada par de trabajos, de modo que la estructura gráfico-teórica se convierta en un árbol binario. Los expertos en la materia apreciarán que son posibles otras implementaciones, por ejemplo, Si el algoritmo pudiera elegir entre un conjunto de determinaciones que permitieran la presencia de retrasos de holgura mínimos entre trabajos en conflicto (es decir, restricciones del tipo $r_i - r_j \geq mi_j + d_{ij}$, dónde d_{ij} es una cantidad positiva), entonces la estructura resultante se convertiría en un *árbol k-ario*, con k igual al número de posibles retrasos de holgura asociados con cada restricción de exclusión mutua.

La organización del algoritmo da espacio a varias optimizaciones posibles que afectan significativamente el rendimiento, como se remarca a continuación.

Elección heurística entre determinaciones de restricciones de exclusión

Dependiendo de la implementación real de las operaciones Enqueue (<D, c>) y Dequeue (), se pueden aplicar varias políticas para la visita del árbol de búsqueda (es decir, la anchura primero, la profundidad primero, etc.). Adoptamos una visita de profundidad primero, que es la elección natural más simple, ya que el objetivo del algoritmo es encontrar, lo antes posible, una hoja asociada a una solución para el problema de programación. La cola Q de enumeración se implementa como una cola LIFO.

El comportamiento del algoritmo también está determinado por el orden en que D_r y D_l se agregan a la cola (es decir, el orden de las líneas 14 y 21 del Alg.2): por ejemplo, si D_l Siempre se agrega primero, se realiza una visita previa al pedido; Y viceversa, se obtiene una visita posterior al pedido agregando siempre D_r primero. En nuestro caso, una buena heurística siempre debería tomar decisiones que probablemente conduzcan a soluciones factibles sin agregar demasiados costos al algoritmo de búsqueda. Presentamos aquí dos heurísticas simples que influyen en el comportamiento del algoritmo de acuerdo con los valores en conflicto de estrechez y *factibilidad*:

- *factibilidad* representa la propiedad de explorar las primeras configuraciones sueltas que probablemente se pueden extender hasta su finalización sin encontrar restricciones inviables. De manera intuitiva, esto puede lograrse seleccionando la restricción de exclusión que deja más grado de libertad a los trabajos restringidos J_i y J_j en la configuración resultante, es decir, aquella para la que la distancia entre los tiempos de liberación $|r_i - r_j|$ es libre de cubrirse en el intervalo más grande (lo denominamos heurística "segura" o "de intervalo más largo");

- La propiedad de estrechez apunta al objetivo opuesto de obtener programaciones compactas, incluso a riesgo de encontrar soluciones más inviables. Como consecuencia, las heurísticas resultantes (lo que denominamos heurísticas "codiciosas" o "de intervalo más corto") seleccionan la restricción que reduce más la distancia entre r_i y r_j.

Simplificación de problemas mediante análisis de intervalos de tiempo

El orden dado a las restricciones de exclusión afecta el rendimiento del algoritmo y condiciona el descarte temprano de determinaciones que no conducen a programaciones completas. Además, algunos refinamientos apuntan a la reducción del número de pares de empleos *C* en conflicto; Si bien este tipo de optimizaciones no afectan la complejidad teórica del algoritmo, en la mayoría de los casos prácticos pueden reducir drásticamente el tiempo de cálculo. Algunas de las opciones en las que se debería agregar la restricción de exclusión en cada etapa del algoritmo ya están resueltas mediante restricciones de D_{prec}; por ejemplo, si dos eventos que requieren el mismo recurso se limitan explícitamente a ocurrir en secuencia, la restricción de exclusión es redundante y se puede ignorar. En la aplicación específica, esto determina una fuerte aceleración, ya que los trabajos representan acciones del sistema que se estructuran naturalmente a lo largo de secuencias lineales. Se puede lograr una reducción más sutil pero altamente efectiva considerando aquellos pares de eventos que, incluso si no están restringidos explícitamente, se asegura que no se superpongan por la configuración de los retrasos. El ejemplo más simple de tal condición está constituido por tres trabajos J_i , J_j y J_k tal que J_i y J_k están obligados a comenzar después de la finalización de J_j, y $d_{ij}^+ + e_i < d_{kj}^-$. La satisfacción de estas tres condiciones, de hecho, garantiza que J_i y J_k no se superpongan incluso si no existe una restricción explícita entre ellos. Son posibles interacciones más complejas, que requieren un mayor esfuerzo para ser identificadas; además, a medida que las restricciones se agregan de manera iterativa durante el procedimiento de

búsqueda, pueden ocurrir dinámicamente condiciones similares en la construcción de zonas D_r y D_l (líneas 8 y 15 del Alg. 2), incluso si no están verificadas en la zona D_{prec} inicial; sin embargo, se puede obtener una detección directa de tales condiciones a través de la observación directa de la forma normal de la zona D extraída de la cola (línea 7 del Alg. 2). A su vez, esto se puede traducir en una optimización del algoritmo de búsqueda que permite la detección temprana de conflictos, evitando la construcción innecesaria y la normalización de las zonas. D_r y D_l.

Reducción de la complejidad mediante el reinicio en caliente del algoritmo Floyd-Warshall

Una segunda optimización logra reducir la complejidad de tiempo asintótica del procedimiento, ya que opera en la estructura de su motor central, el algoritmo Floyd-Warshall.

La formulación estándar del algoritmo Floyd-Warshall utilizado para encontrar la forma normal de las DBM (Alg. 1) se ejecuta en el tiempo $O(N^3)$ con respecto a la dimensionalidad de la zona (en nuestro caso, con respecto al número de trabajos). Sin embargo, la normalización invocada en las líneas 9 y 16 del Alg. 2 presenta una peculiaridad que permite un refinamiento algorítmico específico; de hecho, zonas D_r y D_l se obtienen agregando una única restricción a la zona D extraída de la cola en la etapa 7, que se encuentra en forma normal. Esto implica que solo algunos de los coeficientes de D_r y D_l Se ven afectados por la normalización; de acuerdo con esto, el algoritmo Floyd-Warshall puede reescribirse para ejecutarse en el tiempo $O(N^2)$ (véase el Alg. 3).

ALGORITMO 3: FLOYD-WARSHALL OPTIMIZADO

```
1 begin
2   for k = 0, . . . , N − 1 do
3       if b_kj > b_ki + b_ij then
4           b_kj = b_ki + b_ij;
5           for l = 0, . . . , N − 1 do
6               b_kl = min{b_kl , b_ki + b_ij + b_jl};
7           end
8       end
9   end
10  for h = 0, . . . , N − 1 do
11      b_ih = min{b_ih, b_ij + b_jh};
12  end
13 end
```

Simplificación de problema a través de la agregación de eventos

Se ha introducido una tercera optimización para reducir el tamaño de la matriz N de DBM cuando el problema comprende trabajos con *retrasos inmediatos*, es decir, trabajos que están limitados a ejecutarse inmediatamente uno después del otro. Esto puede suceder si y solo si se establece una restricción de prioridad entre los dos trabajos J_i y J_j tal que $d_{ij} = d_{ij}^+ = d_{ij}^- = 0$.

En este caso los trabajos. J_i y J_j se pueden fusionar en un solo trabajo J_{ij} con tiempo de liberación $r_{ij} = r_j$ y tiempo de ejecución $e_{ij} = e_i + e_j$. Esto resulta en una matriz de DBM reducida en una fila y una columna.

Optimizaciones de código de bajo nivel

El núcleo del algoritmo de búsqueda en el Alg. 2 equivale a solo algunas docenas de líneas de código; para implementar con éxito una versión de producción en la plataforma de destino, se implementó una biblioteca ANSI-C ligera, que ofrece funcionalidades accesorias como la gestión de entrada-salida, control de tiempo de ejecución y conmutación en línea entre diferentes heurísticas, para un total de aproximadamente 2600 líneas de código. Debido a las severas limitaciones de hardware, se han adoptado varias optimizaciones de código además de los refinamientos algorítmicos, para reducir aún más el costo de tiempo y espacio. El más importante de ellos implica la codificación real de soluciones intermedias. En el pseudocódigo del Alg. 2 se usa una cola para enviar las soluciones D intermedias a medida que se generan; Aunque este es el enfoque estándar adoptado en los algoritmos de búsqueda, en el presente caso saturaría fácilmente la memoria disponible. Como solución alternativa, el algoritmo implementado memoriza completamente solo la zona inicial D_{prec} y la zona intermedia real D, que utiliza una pila para codificar una representación compacta de las restricciones de exclusión elegidas hasta ese momento. Siempre que se espere una reversión, el algoritmo reconstruye una nueva solución a partir de D_{prec} De acuerdo con las informaciones encontradas en la pila; en este caso, dado que se modifican los coeficientes múltiples, se requiere una normalización completa (es decir, la del Alg. 1), pero esta pequeña coste se justifica en gran medida por la fuerte reducción en la ocupación del espacio.

La figura 5 es una representación esquemática de la solución de acuerdo con una realización preferida de la presente invención. La búsqueda se formula como un árbol transversal que realiza una serie de decisiones binarias sucesivas sobre restricciones de exclusión. El algoritmo de búsqueda tiene como objetivo encontrar zonas que produzcan soluciones factibles y seleccionar entre dichas soluciones la que corresponda a la solución óptima. Las características clave del presente método son las de permitir la selección heurística de trayectorias "razonables" de acuerdo con

criterios predeterminados para que se pueda encontrar una solución sin explorar todos los casos posibles. En el presente ejemplo, consideramos dos posibles heurísticas: "codiciosa" (o heurística de intervalo más corto) y "segura" (o heurística de intervalo más largo). Las cruces indican un nodo en el árbol desde donde no se pueden encontrar soluciones posibles, por lo tanto, todos los nodos y las hojas que salen de dicho nodo se pueden descartar sin más cálculos.

Las simplificaciones de problemas mencionadas anteriormente alcanzaron los resultados de acelerar el cálculo al reducir la complejidad. En particular, el análisis de intervalo de tiempo minimiza el número de restricciones independientes; el reinicio en caliente reduce la complejidad de la etapa central desde $O(N^3)$ a $O(N^2)$; mientras que la agregación de eventos reduce el número de variables.

Las tablas A y B de la Figura 6 muestran los resultados de una prueba de rendimiento con una comparación del método actual utilizando dos implementaciones diferentes, respectivamente, con una heurística "codiciosa" y con una "segura".

Los resultados experimentales se midieron en un ordenador de escritorio (con procesador de doble núcleo a 2 GHz). Se cree que esto es aproximadamente 10 veces más rápido que un entorno real en una máquina electromecánica como aquella en la que se debería ejecutar el algoritmo actual (es decir, una ejecución en una plataforma de destino tomaría aproximadamente 10 veces más que la misma ejecución en el ordenador de escritorio de la presente prueba).

Ejecutamos nuestro algoritmo de búsqueda en un conjunto de pruebas compuesto por tres Protocolos Analíticos (AP1-AP3), que son ejemplos realistas tomados de la práctica de los análisis biológicos del mundo real:

AP1: 2 secciones, 1 a 3 acciones de pipeta por posición, 4 acciones de cabezal de lectura por sección, retrasos no deterministas;

AP2: 2 secciones, 1 a 3 acciones de pipeta por posición, 4 acciones de cabezal de lectura por sección, retrasos no deterministas;

AP3**:** 2 secciones, 1 acción de pipeta por posición, 4 acciones de cabezal de lectura por sección, retrasos no deterministas;

Los experimentos se realizaron agregando incrementalmente los refinamientos propuestos al algoritmo de búsqueda, que se ejecuta adoptando las heurísticas de intervalo más corto y más largo. Los resultados se muestran en las Tablas A y B de la Figura 6, respectivamente; vale la pena en particular observar cómo se comparan con el cálculo de 20 minutos requerido por el enfoque del Modelo de Comprobación. Como se esperaba, los refinamientos sucesivos del algoritmo producen mejores resultados: en particular, una mayor profundidad del árbol de búsqueda otorga mayor relevancia a la poda realizada a través del análisis de intervalos de tiempo y la aceleración del algoritmo Floyd-Warshall. Otro resultado interesante proviene de la comparación entre heurísticas de intervalo más largo y más corto; en casos de prueba realistas, este último ha mostrado mejores rendimientos que el anterior, tanto en la búsqueda completa como en la convergencia a la mejor solución (en los casos seleccionados, la heurística de intervalo más corto siempre encuentra la mejor solución como la primera).

Para cada protocolo biológico probado (AP1-AP3) y cada nivel de optimización (Básico, TIA, TIA + WR, TIA + WR + EA), el rendimiento se caracteriza como triple primero / mejor / total donde primero es el momento en que se ha encontrado la primera solución, mejor es el momento en el que se ha encontrado la mejor solución y total es el tiempo total de ejecución. TIA es un acrónimo de Análisis de Intervalo de Tiempo, WR es un acrónimo de Reinicio en Caliente y EA es un acrónimo de Agregación de Evento. En las últimas dos columnas, también informamos del intervalo de tiempo de la primera y las mejor soluciones encontradas.

Se apreciará que se pueden hacer alteraciones y modificaciones a lo anterior sin apartarse del alcance de la descripción. Naturalmente, para satisfacer los requisitos locales y específicos, una persona experta en la técnica puede aplicar a la solución descrita anteriormente muchas modificaciones y alteraciones. Particularmente, aunque la presente descripción se ha descrito con un cierto grado de particularidad con referencia a la (s) realización (es) preferida (s) de la misma, debería entenderse que son posibles varias omisiones, sustituciones y cambios en la forma y detalles, así como otras realizaciones; además, se pretende expresamente que los elementos específicos descritos en relación con cualquier realización descrita de la descripción puedan incorporarse en cualquier otra realización como una cuestión general de elección de diseño.

Por ejemplo, se aplican consideraciones similares si los componentes tienen una estructura diferente o incluyen unidades equivalentes. Además, los expertos en la técnica deberían entender fácilmente que la presente invención puede extenderse a.

REIVINDICACIONES

1. Un método de programación, en un sistema adaptado para ejecutar una pluralidad de trabajos no preferibles J_1 .... J_n, con tiempos de ejecución deterministas e_1 .... e_n, cada trabajo está asignado estáticamente a una de m máquinas independientes M_1 .... M_m, para determinar un tiempo de liberación r_1 .... r_n para cada trabajo J_1 .... J_n a fin de minimizar el tiempo de finalización del conjunto general de trabajos, respetando la siguiente restricción de precedencia:

- para un conjunto predeterminado de pares J_x y J_y de la pluralidad de puestos de trabajo, el retraso entre la finalización de J_y y el comienzo de rango de J_X dentro de un retraso mínimo d^-_{xy} y un retraso máximo d^+_{xy}:

$$d^-_{xy} \leq r_x - (r_y + e_y) \leq d+_{xy}$$

y las siguientes restricciones de exclusión:

- para un conjunto predeterminado de pares J_i y J_k de la pluralidad de trabajos, se cumple una de las siguientes dos determinaciones de restricción de exclusión:

$$r_i \geq r_k + e_k \text{ o } r_i + e_i \leq r_k$$

de forma que los periodos de ejecución de J_k y J_i no se superponen,

el método que comprende las etapas de:

- A) construir un conjunto fronterizo que incluya una zona de Matriz de Límite de Diferencia (DBM) que recopila tiempos de liberación que satisfacen las restricciones de precedencia;

- B) seleccionar una de las zonas de DBM y eliminarla del conjunto de fronteras:

- C) responder a la zona de DBM seleccionada y no satisfacer al menos una restricción de exclusión de un par J_i y J_k del conjunto predeterminado de pares, construyendo una zona restringida de la zona de DBM seleccionada por cada una de las dos determinaciones que resuelven el conflicto ($r_i \geq r_k + e_k$ o $r_i + e_i \leq r_k$);

- D) verificar si las zonas de DBM restringidas no están vacías y agregar las zonas de DBM restringidas no vacías al conjunto de fronteras;

- E) repetir las etapas B a D hasta que una zona de DBM seleccionada cumpla con todas las restricciones de exclusión.

2. El método de la reivindicación 1, en el que la etapa de repetición se realiza hasta que todas las zonas de DBM en el conjunto de fronteras satisfacen todas las restricciones de exclusión.

3. El método de la reivindicación 2, en donde la etapa de repetición se detiene cuando ha sido alcanzado un tiempo de búsqueda máximo predeterminado.

4. El método de cualquiera de las reivindicaciones anteriores que además incluye:

- seleccionar una programación óptima dentro de todas las zonas de DBM que satisfaga todas las restricciones de exclusión.

5. El método de planificación de cualquiera de las reivindicaciones anteriores que en el que la etapa B se realiza basándose en una estimación heurística.

6. El método de planificación de la reivindicación 5, en el que la heurística incluye una heurística de intervalo más largo (segura), que estima la capacidad de la zona de DBM seleccionada de satisfacer todas las restricciones de exclusión.

7. El método de planificación de la reivindicación 5 o 6, en el que la heurística incluye una heurística de intervalo más corto (codiciosa), que estima la capacidad de la zona de DBM seleccionada para minimizar el tiempo de finalización general.

8. El método de planificación de cualquiera de las reivindicaciones anteriores que, en donde, en la ejecución de la etapa C, la forma normal de las zonas restringidas se deriva utilizando una versión optimizada del algoritmo Floyd-Warshall que reduce la complejidad de $O(n^3)$ a $O(n^2)$, donde n es el número de trabajos, evitando el recálculo de coeficientes de DBM que no se ven afectados por la restricción

9. El método de planificación de recálculo en donde se implementan optimizaciones de código para limitar la complejidad temporal del algoritmo y la ocupación de espacio de las estructuras de datos empleadas.

10. El método de recálculo, en donde el sistema incluye una pluralidad de recursos, cada trabajo que requiere la disponibilidad de al menos uno de la pluralidad de recursos para la ejecución, la restricción de exclusión entre cada par de trabajos predeterminado representa el requisito de disponibilidad del mismo recurso Por ambos trabajos del

par .

11. El método de la reivindicación 10, en donde los recursos incluyen cualquier unidad lógica, electrónica o mecánica empleada por cualquiera de los trabajos programados para la evaluación de su tarea.

12. Un programa informático, que incluye medios de código de programa, para realizar el método de programación de recálculo cuando el programa se ejecuta en un ordenador.

13. Un medio legible por ordenador que almacena el programa informático de la reivindicación 12.

14. Un sistema para programar la ejecución de una pluralidad de trabajos no preferibles J_1 .... J_n, con tiempo de liberación r_1 .... r_n y tiempos de ejecución deterministas e_1 .... e_n, cada trabajo está asignado estáticamente a una de m máquinas independientes M_1 .... M_m, el sistema que incluye uno o más componentes adaptados para realizar las etapas del método de programación de cualquier reivindicación 1 a 11.

15. Un sistema electromecánico para realizar análisis biológicos, que incluye el sistema de la reivindicación 14.

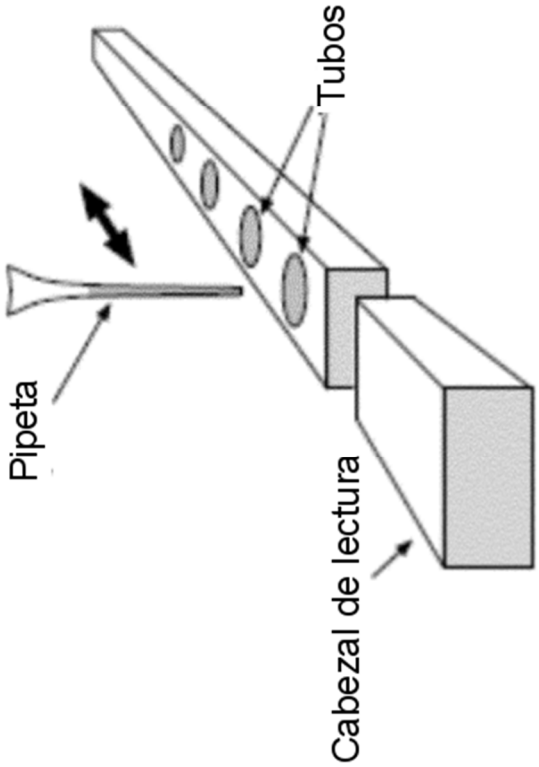

Fig. 1

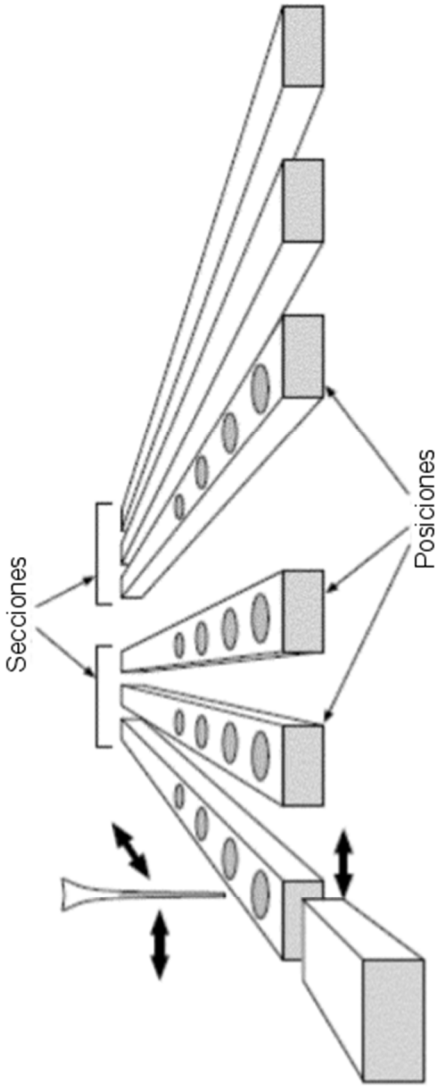

Fig. 2

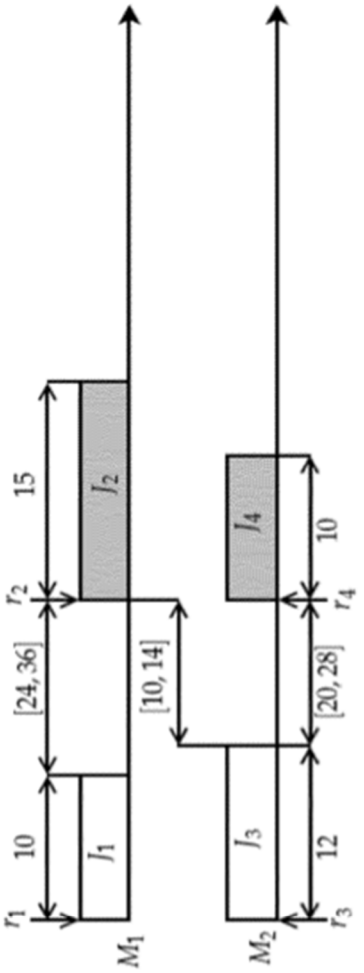
r_1
10
J_1
M_1
[24, 36]
15
J_2
r_2
[10, 14]
J_3
M_2
12
r_3
[20, 28]
J_4
10
r_4

Fig. 3a

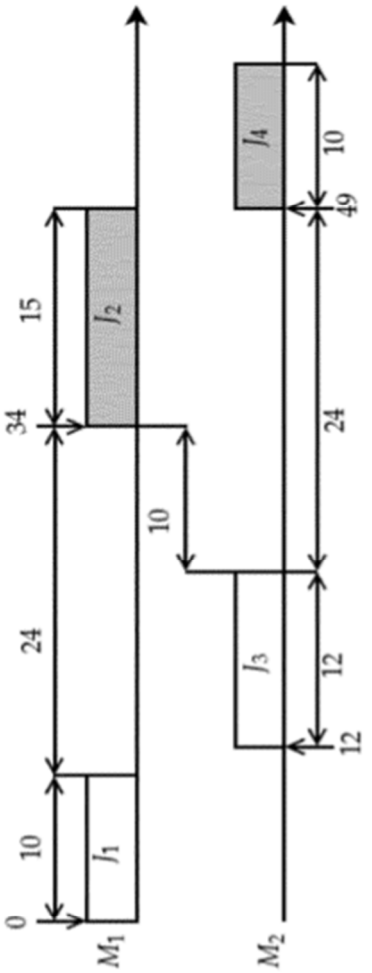
0
10
J_1
M_1
24
34
15
J_2
10
J_3
12
12
M_2
J_4
49
10
24

Fig. 3b

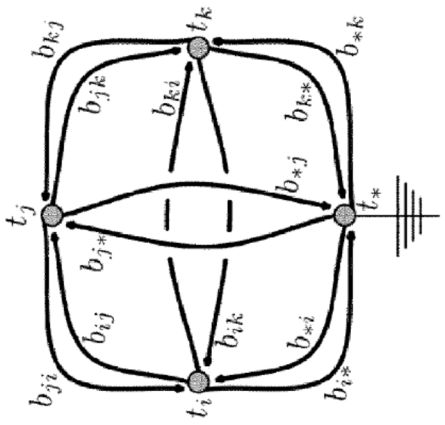

Fig. 4c

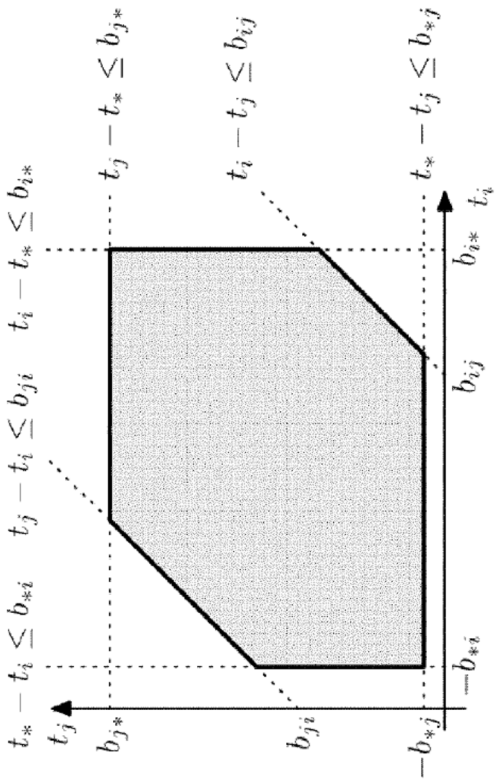

Fig. 4a

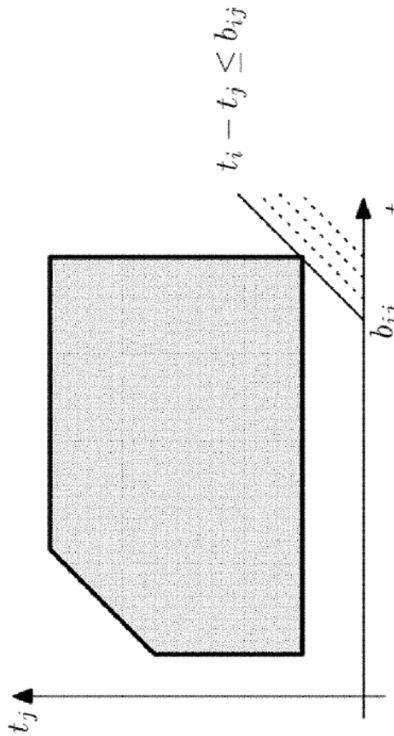

Fig. 4b

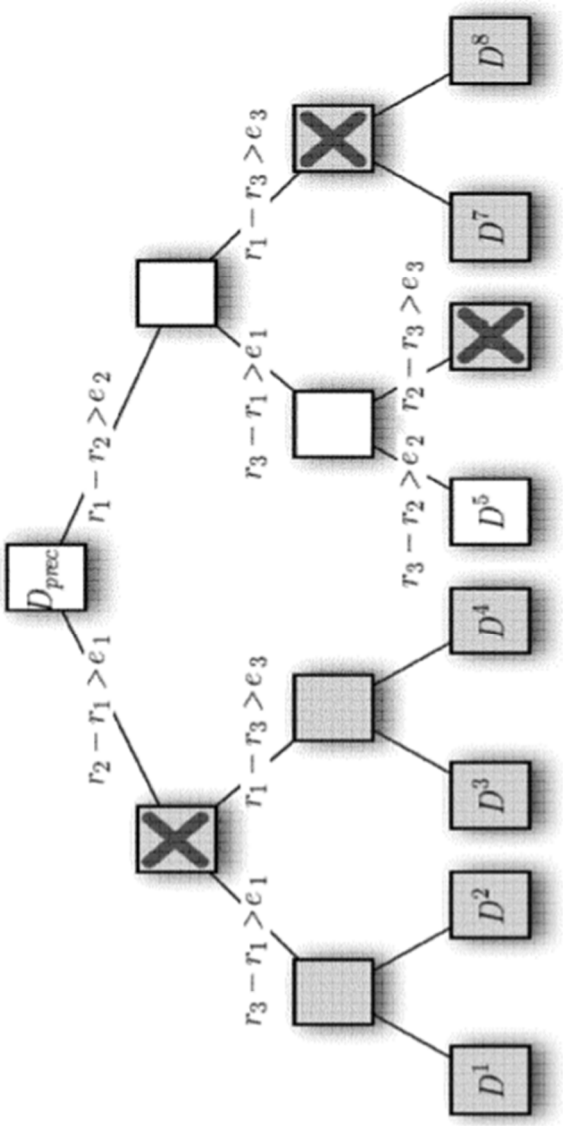
D_{prec}
$r_2 - r_1 > e_1$
$r_1 - r_2 > e_2$
$r_3 - r_1 > e_1$
$r_1 - r_3 > e_3$
$r_3 - r_1 > e_1$
$r_1 - r_3 > e_3$
$r_3 - r_2 > e_2$
$r_2 - r_3 > e_3$
D^1
D^2
D^3
D^4
D^5
D^7
D^8

Fig. 5

	Poda (ms)	TIA Poda (ms)	TIA+ WR Poda (ms)	TIA + WR + EA Poda (ms)	Primero	Mejor
BP1	52/52/66	36/36/51	20/20/25	1/1/16	25221	25221
BP2	43/43/43	34/34/34	25/25/25	1/1/20	25221	25221
BP3	72/72/72	65/65/65	17/17/17	1/1/15	2461	24461

(A)

	Poda (ms)	TIA Poda (ms)	TIA + WR Poda (ms)	TIA + WR + EA Poda (ms)	Primero	Mejor
BP1	71/83/83	50/62/62	31/51/51	1/2/40	32115	25221
BP2	73/107/107	42/53/53	31/42/42	1/2/20	32875	25221
BP3	80/84/84	45/52/52	32/39/39	1/2/25	32115	24461

(B)

Fig. 6