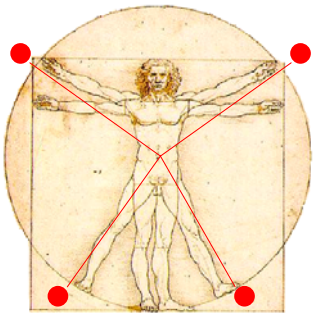
TECNOLOGÍ@ y *DESARROLLO*

Revista de Ciencia, Tecnología y Medio Ambiente

VOLUMEN V . AÑO 2007

SEPARATA



METAHEURÍSTICAS DE OPTIMIZACIÓN COMBINATORIA: USO DE SIMULATED ANNEALING PARA UN PROBLEMA DE CALENDARIZACIÓN

Pilar Moreno Díaz, Gabriel Huecas Fernández-Toribio, Jesús Sánchez Allende, Almudena García Manso.

UNIVERSIDAD ALFONSO X EL SABIO
Escuela Politécnica Superior

Villanueva de la Cañada (Madrid)

ISSN: 1696-8085
Editor: Julio Merino García tecnologia@uax.es

METAHEURÍSTICAS DE OPTIMIZACIÓN COMBINATORIA: USO DE SIMULATED ANNEALING PARA UN PROBLEMA DE CALENDARIZACIÓN

Pilar Moreno Díaz[a], Gabriel Huecas Fernández-Toribio[b], Jesús Sánchez Allende[a], Almudena García Manso[a]

[a)]Departamento de Electrónica y Sistemas, Escuela Politécnica Superior. Universidad Alfonso X el Sabio. Avda. de la Universidad nº1, Villanueva de la Cañada, 28691 Madrid. España. Tlf.:918109200, email: pilar@uax.es

[b)]Departamento de Ingeniería de Sistemas Telemáticos. Escuela Técnica Superior de Ingnieros de Telecomunicación. Universidad Politécnica de Madrid. 28040 Madrid. España. e-mail: gabriel@dit.upm.es

RESUMEN: En el mundo real existen multitud de problemas cotidianos que, desde un punto de vista ingenieril, precisan de una solución que cumpla un conjunto de requisitos de la manera más apropiada maximizando o minimizando determinado valor. Sin embargo, los problemas con los que nos enfrentamos pertenecen a la clase NP-duros o, incluso, a la clase NP-completos. Las formas de abordar la obtención de soluciones factibles para su aplicación práctica son variadas. Una de las estrategias que ha obtenido una gran aceptación y que ha ido consiguiendo un cuerpo formal importante es la metaheurística llamada Simulated Annealing. En este articulo se presentan las características más importantes de esta metaheurística, sus principales resulados teóricos y la aplicación a un problema de calendarización que está resultando de aplicación en la Universidad Alfonso X el Sabio.

PALABRAS CLAVE: Optimización combinatoria, complejidad algorítmica, Simulated Annealing, calendarización, problemas NP-completos. Problemas NP-duros.

ABSTRACT: There are lots of usual problems in the real world which makes them hard to obtain a soluticon with a maximization or minization of a function value. Real problems demonstrate to be usually in the class of NP-hard problems or even in the class NP-complete. To obtain a reasonable solution the only way is to use some heuristic or metaheuristic aproach. One of them is Simulated Annealing. This paper presents the most important caracteristics of Simulated Annealing, its main theoretical results and the application for a timetabling problem: the examination timetable of a course for Universidad Alfonso X el Sabio.

KEY-WORDS: Combinatorial optimization, algorithm complexity, Simulated Annealing, NP-complete. NP-hard.

http://www.uax.es/publicaciones/archivos/TECELS07001.pdf

1. Introducción

En la vida cotidiana nos encontramos con multitud de problemas que exigen dar una solución. Algunos de los que se pueden citar incluyen:

- Crear un plan de *mínimo coste* para repartir mercancías a clientes
- Realizar una *asignación óptima* de trabajadores a un conjunto de tareas
- Encontrar una *secuencia óptima* de trabajos en una cadena de producción
- Encontrar una distribución de tripulaciones de aviones con *mínimo coste*
- Encontrar la *configuración óptima* en una red de telecomunicaciones.
- Crear un calendario de exámenes que *minimice* la probabilidad de solapamientos, etc.

Todos ellos son problemas que presentan una característica común, tratan de optimizar (maximizar o minimizar) un determinado valor. Por ello se llaman problemas de optimización.

En general un problema de optimización supone que se desea maximizar o minimizar una determinada función de una o más variables:

$$y = f(x) \quad \text{(Ec. 1.1)}$$

donde

$$x \in X, x = \{x_1, x_2, \ldots, x_n\} \quad \text{(Ec. 1.2)}$$

siendo x un vector en el que cada una de las variables x_i definen, en cierto sentido parte de la solución. De hecho, a cada uno de estos vectores x, se les denomina *vector solución* o vector candidato a solución.

En este esquema se denomina *espacio de búsqueda* al conjunto de vectores posibles que existen para un problema determinado. A la función f se le denomina *función objetivo*. Cuando se trata de maximizar se suele hablar de valor y cuando se trata de minimizar se suele hablar de coste. De esta forma el objetivo será o bien maximizar el valor o bien minimizar el coste.

Sin embargo, en este esquema general, los problemas de optimización tienen formas muy variadas y cada una con características muy diferentes. La variación en las

características de los problemas afecta de forma importante a los modelos de resolución de los mismos, por ellos se han creado muchos métodos que intentan abordar el proceso de optimización.

En Inteligencia Artificial se suele denominar el calificativo de *heurístico* a cualquier método que tenga en cuenta el empleo del conocimiento específico del problema para su resolución. En este sentido (Reeves, 1995) define un *algoritmo heurístico* como un procedimiento de búsqueda de soluciones cuasi-óptimas a un coste computacional razonable, sin ser capaz de garantizar la optimalidad de las soluciones empleadas ni determinar a qué distancia de la solución óptima nos encontramos.

Esta línea de investigación ha contribuido a la creación de métodos y procedimientos que implican estrategias generales de resolución de problemas. Las *metaheurísticas* (Glover, 1986) son estrategias inteligentes generales para diseñar o mejorar procedimientos heurísticos para la resolución de problemas con un alto rendimiento (Melián, 2003).

La técnica de Simulated Annealing pertenece a la familia de metaheurísticas llamadas Generalizad Ramdomized Hillclimbing (Jacobson, 2004).

2. Problemas y complejidad

Cuando uno trata con la resolución de problemas rápidamente se da cuenta de que no todos ellos presentan el mismo grado de dificultad. De forma que dado un problema cualquiera, ¿cómo se puede decir si es fácil o difícil? O más aún, ¿qué significa que un problema sea fácil o difícil? De este tema trata una rama de las matemáticas denominada *complejidad algorítmica*. La complejidad algorítmica establece una clasificación de los distintos tipos de problemas por su grado de dificultad de acuerdo con la complejidad computacional del algoritmo más sencillo que permite asegurar su resolución.

A grandes rasgos los problemas se pueden clasificar en dos grandes conjuntos: los tratables y los intratables. Los problemas intratables incluyen los que son formalmente indecidibles (Minsky, 1967), para los que existe una demostración de que no existe ningún algoritmo que permita resolverlos en todos los casos. Los problemas intratables también incluyen todos aquellos problemas para los que sí se conoce un algoritmo que podría resolverlos pero para los que la cantidad de tiempo computacional necesaria para

hacerlo los convierte realmente en inabordables, incluso para tamaños "razonables" de los mismos. Y ello, es independiente de la capacidad computacional de que se disponga. Formalmente se puede decir que no existe ningún algoritmo que permita resolverlo en un número de pasos que sea una función polinomial del tamaño de entrada del problema.

Un problema tratable, un problema de la *clase P*, es un problema que se puede resolver siempre utilizando un algoritmo que conlleva un numero de pasos que es función polinomial del tamaño de entrada del problema.

En resumen, se puede decir que los problemas de la clase P se pueden resolver en tiempo polinomial y los intratables no se pueden resolver en tiempo polinomial.

Adicionalmente, se puede establecer una clasificación adicional para aquellos problemas decidibles pero intratables, para los que al menos existe la posibilidad de calcular, en un número de pasos que es función polinomial del tamaño del problema, si una solución pertenece a las soluciones del mismo. Estos problemas junto con los de la clase P forman la clase NP. Los problemas de la clase NP son los que se pueden resolver utilizando una máquina imaginaria llamada máquina de Turing no determinista (NDTM, Non-Deterministic Turing Machine), en un número de pasos polinomial.

Muchos de los problemas de la clase NP son problemas muy comunes, que aparecen de forma habitual en distintos áreas de la ingeniería e incluyen, entre otros, problemas de partición de conjuntos, diseño de redes, planificación, optimización, recuperación de información, etc (Garey, 1979).

De todos los problemas de la clase NP se puede distinguir un conjunto de ellos denominado NP-completos, que son los más difíciles de resolver. El Teorema de Cook (Cook, 1971) nos permite determinar si un determinado problema NP pertenece a la clase NP-completo. La propiedad de la clase NP-completo es que todo problema de la clase NP se puede transformar polinomialmente en él.

Sin embargo, los problemas de optimización no se encuentran, en general, en la clase NP y, por tanto, puede que no sean NP-completos. Es así porque, en general, no es posible comprobar si se ha conseguido una solución óptima en un número de pasos que

sea función polinomial del tamaño del problema. En la mayoría de los casos sólo es posible comprobarlo comparándola con todo el conjunto de soluciones del problema. Si el conjunto de soluciones crece exponencialmente con el tamaño del problema resulta evidente que la comprobación no se puede llevar a cabo en tiempo polinomial. Estos problemas de optimización se encuentran en una clase de problemas que se denominan NP-duros.

De todo lo dicho anteriormente resulta indiscutible que para los problemas de optimización NP-duros no existe ningún algoritmo en tiempo polinomial que permita determinar la solución óptima al problema. Por ello se utilizan *métodos aproximados* mediante heurísticas que permiten aproximarse a una solución óptima, generando soluciones factibles al problema que resulten de utilidad práctica.

3. Simulated Annealing

3.1. Introducción

Las metaheurísticas son procedimientos generales que permiten generar soluciones aproximadas a problemas. Existen muchos tipos de metaheurísticas, pero en general se podrían clasificar en (Melián, 2003):

- *Metaheurísticas de relajación*: son procedimientos que relajan las condiciones del problema original, es decir, modifican el problema de forma que consiguen que el problema original sea más fácil de resolver.
- *Metaheurísticas constructivas*: se orientan a procedimientos que obtienen una solución al problema a partir del análisis y selección paulatina de las componentes que la forman. Los algoritmos más utilizados son los algoritmos voraces (*greedy algorithms*) que construyen soluciones buscando lo mejor de partes de la misma sin buscar óptimos globales.
- *Metaheurísticas de búsqueda*: utilizan procedimientos mediante transformaciones o movimientos para recorrer el espacio de soluciones y explotar las estructuras de entornos asociadas. Entre ellas se encuentran los algoritmos de Hillclimbing, de arranque múltiple (Martí, 2003), Simulated Annealing (Kirkpatrick, 1983), búsqueda tabú (Glover, 1997), etc.
- *Metaheurísticas evolutivas*: tratan el problema utilizando conjuntos de soluciones que evolucionan sobre el espacio de soluciones. Entre ellas se encuentran los algoritmos genéticos (Holland, 1975), meméticos (Moscato,

2003), estimación de distribuciones (Lozano, 2001), reencadenamiento de caminos (Glover, 2003), colonias de hormigas (Dorigo, 1996), etc.

Asimismo se desarrollan metaheurísticas híbridas que en el método de obtención de soluciones utilizan dos o más de las metaheurísticas anteriores. Así, por ejemplo, se puede utilizar una metaheurística de algoritmos genéticos para encontrar una solución factible al problema y a continuación utilizar una metaheurística de Simulated Annealing para mejorar la solución.

El número de metaheurísticas que se puede recoger en la literatura es muy amplio generándose continuamente nuevas metaheurísticas que aparecen en las revistas especializadas como puede ser el Journal of Heuristics o un conjunto cada vez más numeroso de libros dedicados al tema (Michalewicz, 2000), (Oates, 2000), (Laguna, 2000), (Ribeiro, 2001), (Glover, 2003). Sin embargo, en muchos casos los investigadores se centran en describir el comportamiento de las nuevas metaheurísticas y comparar su rendimiento con otras conocidas sin centrarse en intentar explicar las propiedades que hacen que el comportamiento de las mismas sea apropiado ni sus condiciones de aplicabilidad.

De este conjunto de metaheurísticas está teniendo gran aceptación las que conforman el grupo de metaheurísticas de Hillclimbing generalizadas (Jacobson, 2004), para las que se han descrito las condiciones necesarias y suficientes de convergencia al conjunto de soluciones óptimas. Dentro de esta familia de metaheurísticas es de destacar la de Simulated Annealing.

De hecho, de acuerdo con (Rajasekaran, 1992) los resultados empíricos del uso de Simulated Annealing para ciertos algoritmos NP-duros, incluyendo algunos NP-completos demuestran que este tipo de algoritmos consigue mejores resultados que otro tipo de heurísticas, pero en el caso peor, el tiempo utilizado puede ser exponencial.

3.1. Características generales de la metaheurística Simulated Annealing

Este tipo de algoritmos aparece a principios de los años 80 (Kirpatrick, 1983). Se trata de un tipo de modelo de resolución para la optimización de problemas de tipo combinatorio con mínimos locales. Su aproximación consiste en generar aleatoriamente una solución cercana a la solución actual (o en el entorno de la solución) y la acepta

como buena si consigue reducir una determinada función de coste, o con una determinada probabilidad de aceptación. Esta probabilidad de aceptación se irá reduciendo con el número de iteraciones y está relacionada también con el grado de empeoramiento del coste.

Este tipo de algoritmos se diferencia de los algoritmos de Hillclimbing en que en estos últimos solo se acepta una nueva solución si ésta mejora la anterior de manera que cada solución que se obtenga vaya obteniendo soluciones mejores cada vez. El problema de este tipo de algoritmos es que la solución final suele encontrarse en un mínimo local no explorando con suficiente amplitud el espacio de soluciones. Sin embargo, en la aplicación del algoritmo de enfriamiento lento se pueden aceptar soluciones que empeoran la solución actual, solo que esta aceptación dependerá de una determinada probabilidad que depende de un parámetro, denominado temperatura, del estado del sistema.

Estos algoritmos se llaman de esta forma por su parecido en funcionamiento al proceso de enfriamiento metalúrgico: cuando se enfría un metal fundido suficientemente despacio, tiende a solidificarse en una estructura de mínima energía (equilibrio térmico); según va disminuyendo la temperatura, las moléculas del metal tienen menor probabilidad de moverse de su nivel energético. La probabilidad de movimiento de una molécula de su estado viene dada por la función de Boltzmann $P(\Delta E) = e^{-\Delta E/kT}$, que depende de la temperatura, de la diferencia de energía y de una constante K (la constante de Boltzmann).

Los algoritmos de enfriamiento lento simulado tienen algunas ventajas con respecto a otras técnicas de optimización global. De acuerdo con (Elmohamed, 1998) entre las ventajas de estos algoritmos se pueden citar:

- Lo relativamente sencillo que resulta implementar este tipo de problemas.
- Su aplicabilidad a la mayoría de los problemas de optimización con una estructura combinatoria.
- Su capacidad para ofrecer soluciones razonablemente buenas a la mayoría de los problemas, aunque hay una cierta dependencia de la planificación del enfriamiento y los movimientos que se realicen.
- La facilidad con la que se puede combinar este tipo de algoritmos con otras técnicas heurísticas como los sistemas expertos, los algoritmos genéticos, las

redes neuronales, etc, consiguiendo sistemas híbridos que pueden resultar de gran potencia en la resolución de problemas muy complejos.

Por otra parte, también se pueden citar algunos aspectos que pueden limitar su utilización:

- Se necesita elegir con mucho cuidado los movimientos que se realizan, así como los parámetros que se van a utilizar para tratarlo, como por ejemplo la tasa de enfriamiento.
- Una ejecución del problema puede requerir mucho tiempo de cálculo.
- Puede que sea necesario realizar muchas ejecuciones para encontrar una solución satisfactoria.
- Dependiendo de los parámetros elegidos, las soluciones que se van encontrando pueden ser poco estables, "saltando" mucho de unas a otras sin encontrar una solución buena con la rapidez suficiente lo que obliga a retocar los parámetros con las distintas ejecuciones.

3.2. Estructura del algoritmo

La estructura básica de un algoritmo de enfriamiento lento simulado se puede escribir de acuerdo con el siguiente seudocódigo:

```
Seleccionar solución inicial s0
Seleccionar temp. inicial t0 > 0
Repetir
  Repetir
    Seleccionar un s∈Ss0
    d ← f(s) – f(s0)
    Si (d < 0) ó (U(0,1) < exp(-d/t))
      s0 ← s
  Hasta un número de repeticiones
  t ← a(t)
Hasta condición de parada
Solución: la mejor de todas las s0 encontradas
```

En este seudocódigo se pueden diferenciar dos tipos de elementos que permiten afinar el algoritmo para que la ejecución del mismo sea la más eficaz de acuerdo con el problema que se desea resolver:

- Elementos genéricos. Son elementos que no tienen una dependencia directa del problema, aunque hay que afinarlos para el problema concreto que se desea resolver.
  - Temperatura inicial: t_0
  - Proceso de enfriamiento: $\alpha(t)$
  - Número de repeticiones.
  - Condición de parada.
- Elementos dependientes de problema. Son los elementos que definen de forma directa el problema que se está resolviendo. Definen un modelo del problema y la estructura del espacio de soluciones para el mismo.
  - Espacio de soluciones: S
  - Estructura de vecindad: S_i
  - Función de coste: f(s)
  - Solución inicial: s_0

Estos elementos se describen con más detalle en las siguientes secciones.

3.2.1. Temperatura inicial

La temperatura inicial define el momento en el que se inicia el proceso de enfriamiento. Esta temperatura inicial debe ser suficientemente alta para permitir el libre cambio de soluciones vecinas y, sobre todo, que la solución final sea independiente de la solución inicial.

Una regla práctica que se le suele poner a la temperatura inicial es que debe tener un valor tal que la proporción inicial de vecinos aceptados (tanto de mejora como de no mejora) tenga un alto valor (90% - 95%). Es decir, inicialmente todos los vecinos se aceptan con una probabilidad cercana a 1.

Lógicamente hay que elegir esta temperatura con cuidado porque si se elige una temperatura inicial demasiado alta se perderá tiempo realizando muchos movimientos

que se aceptan siempre, lo que no resulta de mucho interés pues no se está persiguiendo ningún valor concreto.

Por otra parte, si se elige una temperatura inicial demasiado baja el proceso se mueve poco desde la solución inicial por lo que se puede quedar atrapado desde un inicio en una región desde la que sólo se pueda llegar a un óptimo local, por lo que se perderá la posibilidad de explorar otras regiones donde se encuentre alguno de los óptimos globales del sistema.

3.2.2. Proceso de enfriamiento

El proceso de enfriamiento es el mecanismo por el que la temperatura va tendiendo a 0. Es decir, la probabilidad de aceptación de soluciones peores tiende a 0. El proceso de enfriamiento, por tanto, determina cómo se modifica la temperatura durante la ejecución del algoritmo.

Hay dos modelos básicos que se pueden utilizar para el proceso de enfriamiento. En el primero podríamos ir enfriando muy lentamente con cada iteración de la cadena de Markov de manera que en un número de pasos extremadamente alto la temperatura tienda a 0, ó utilizar un proceso de enfriamiento más rápido, pero antes de aplicar un cambio en la temperatura hay que dejar que el sistema llegue a un estado estacionario, es decir, un estado en el que se estabilice la solución.

Algunos de los algoritmos de enfriamiento utilizados son los siguientes:

- Descensos constantes de temperaturas.
- Descenso exponencial: $t_{k+1} = \alpha\ t_k$, $(0{,}8 < \alpha < 0{,}99)$
- Criterio de Boltzmann: $t_k = T_0/(1 + \log(k))$
- Esquema de Cauchy: $t_k = T_0 /(1+k)$

Hay que intentar durante el proceso de enfriamiento que haya una alta aceptación al principio (de forma que se pueda realizar un proceso de exploración inicial en el espacio de búsqueda) e ir reduciendo esta aceptación hacia el final (de forma que se fomente el proceso de explotación de vecindad para llegar a una solución lo mejor posible).

3.2.3. Número de repeticiones

El número de repeticiones del bucle interno nos va a dar el número de vecinos que se visitan antes de reducir la temperatura del sistema. Es proceso debe ser dinámico, de forma que se aumente el número de repeticiones según se reduce la temperatura. De esta forma se conseguirá explotar la vecindad hacia el final del proceso.

Un mecanismo habitual para conseguirlo es repetir el bucle hasta que se cumpla una de las siguientes condiciones:

- Se aceptan un cierto número de soluciones. Durante las primeras fases del proceso se aceptarán muchas soluciones mientras que en las últimas fases se aceptarán muy pocas. De esta forma según se desarrolla el algoritmo cada vez se analizan más vecinos.
- Se han visitado un cierto número de vecinos. De esta forma el bucle interno termina en algún momento, si se da el caso que se aceptan muy pocos vecinos, o ninguno, de acuerdo con la condición anterior.

Para que se pueda explorar con cierta completitud todos los vecinos el número de repeticiones debe ser del orden del tamaño de la vecindad.

3.2.4. Condición de parada

Esta condición nos indica cuando damos por terminado el algoritmo de cálculo. Hay que tener en cuenta que al reducirse la temperatura, la probabilidad de aceptar soluciones peores tiende a 0. Así mismo, la probabilidad de encontrar soluciones mejores también tiende a 0.

En este sentido las condiciones para terminar el algoritmo deberían tener en cuenta que ya se han realizado un determinado número de iteraciones sin mejorar la solución. La idea es evitar seguir insistiendo reiteradamente en la vecindad de una solución para la que ya se lleva tiempo intentado explotar.

3.2.5. Espacio de soluciones

La definición del modelo del problema determinará la estructura del espacio de soluciones y, por tanto, la forma en que se va a poder recorrer, los algoritmos que se pueden utilizar para crear soluciones, etc.

Resulta de gran importancia que el modelo elegido para crear el modelo del problema disponga, por comodidad algorítmica, de algunas propiedades como: el espacio de soluciones no genere soluciones degeneradas, no permita la generación de soluciones imposibles, la estructura del espacio de costos no sea excesivamente escarpado o llano, etc.

Lógicamente para cada problema hay que estudiar de forma diferenciada cómo generar este espacio de soluciones, en relación con la facilidad para generar una vecindad y el cálculo de la función de coste asociada a una determinada solución.

3.2.6. Estructura de vecindad

Dada una solución, hay que poder conseguir una nueva solución mediante un "pequeño" cambio en la solución original. Además la forma de generar una vecindad nos debe poder asegurar que a partir de una solución cualquiera se debe poder llegar a cualquier otra ya sea directa o indirectamente. Esta característica es vital para dar validez al proceso de convergencia del algoritmo.

Asimismo, la vecindad de una solución dada debería ser suficientemente pequeña como para poder realizar una búsqueda en pocas iteraciones, como suficientemente grande para generar mejoras sustanciales en pocos movimientos.

Desde el punto de vista computacional la generación de un vecino debería ser rápido, de forma que se puedan calcular un mayor número de vecinos por unidad de tiempo. Así mismo, sería muy apropiado que el vecino generado siempre sea una solución factible, lo que puede resultar complicado. Si no es factible conseguir vecinos factibles, se puede permitir que durante el proceso del algoritmo se generen soluciones imposibles, pero éstas se penalicen suficientemente como para que no se elijan más que de forma temporal.

3.2.7. Función de coste

La función de coste mide la bondad de una solución. La función de coste es una medida asociada a todos los elementos de una determinada solución. Uno de los objetivos que deben guiar el desarrollo de la función de coste es que esta sea rápida de calcular al generar una solución de la vecindad, creando un cálculo incremental que tenga en cuenta sólo los cambios que generan la vecindad. De esta forma se podrán calcular más soluciones por unidad de tiempo. De hecho, el cálculo de la función de coste suele ser uno de los aspectos más costosos en tiempo de cómputo del algoritmo.

Asimismo hay que evitar generar una topografía muy plana o muy abrupta. Si la topografía es muy plana, al generar vecinos el coste no variará para muchos de ellos, no permitiendo diferenciar soluciones. En el caso contrario, si la topografía es muy abrupta, la diferencia de coste entre una solución y un vecino de la misma será demasiado grande no permitiendo guiar el proceso de convergencia hacia una solución óptima.

3.2.8. Solución inicial

La solución inicial, en general es cualquier solución válida del problema. Al ser un problema de optimización combinatoria, el modelo permite expresar una solución como una combinación de partes del mismo. De esta forma, la solución inicial más sencilla resulta de elegir una combinación aleatoria de elementos de problema que conformen una solución con la estructura del modelo elegido.

Otra alternativa es utilizar un proceso doble. En primer lugar se emplea algún tipo de heurística que nos permita generar una solución razonablemente buena, mejor que utilizar una solución estrictamente aleatoria.

3.3. Teorema de convergencia

Existen numerosos artículos que tratan sobre la convergencia del algoritmo Simulated Annealing y las condiciones necesarias y suficientes para que se dé el proceso de convergencia. Entre ellos uno de los más importantes en este sentido es el de (Aarts, 1997). En él se demuestra que el proceso converge al óptimo global siempre que la secuencia de nuevas soluciones (cadenas de Markov) se aproxime a una distribución estacionaria.

Asimismo se pueden tener en cuenta los trabajos de (Cohn, 1999) en el que se realiza un análisis a temperatura fija, (Theodospoulos, 1999) en el que se muestra que el uso de soluciones iniciales aleatorias puede mejorar el proceso de convergencia. Villalobos (2004), en el que se analiza la convergencia asintótica en problemas multiobjetivo, etc. Una revisión bastante completa se puede ver también en (Aarts, 2002).

En (Jacobson, 2004) se analiza que las condiciones necesarias y suficientes para que el proceso converja al conjunto de soluciones óptimas son (sólo se enuncian por brevedad):

$$\sum_{k=1}^{\infty} r(k) = \infty \qquad \text{(Ec. 3.1)}$$

$$P\{C^c(k) \mid B^c(k-1)\} \to 0, k \to \infty \qquad \text{(Ec. 3.2)}$$

donde

$$r(k) = P\{B^c(k) \mid B(k-1)\} = P\{C(k) \mid B(k-1)\} \text{ (Ec. 3.3)}$$

es decir, r(k) es la probabilidad de alcanzar un estado global en la macroiteración k-ésima, dado que no se ha visitado ninguno en las k-1 macroiteraciones anteriores. La Ecuación 3.1 indica que no converja a 0 demasiado rápidamente cuando k tiende a infinito. La Ecuación 3.2 quiere decir que la probabilidad condicional de que el algoritmo visite un elemento del conjunto de soluciones óptimas tras haber visitado uno tienda a 1 cuando las macroiteraciones tienden a infinito.

En cuanto a la aplicabilidad de las ecuaciones anteriores estas condiciones necesarias y suficientes de convergencia se deriva para Simulated Annealing (Hajek, 1988) que:

$$\sum_{k=1}^{\infty} e^{-(d/t(k))} = \infty \qquad \text{(Ec. 3.4)}$$

donde d es la altura máxima en el recorrido entre dos mínimos de la función de coste y t(k) es un proceso de enfriamiento que tiende a 0 cuando k tiende a infinito, es una condición necesaria de convergencia.

4. Problema de elaboración de calendarios de exámenes

El problema de calendarización es uno de los problemas clásicos que se utiliza para estudiar los algoritmos heurísticos con estructura combinatoria como los algoritmos de

Simulated Annealing. De forma simplificada consiste en asignar un conjunto de recursos (profesores, aulas, asignaturas, etc) de manera que cumplan un conjunto de restricciones relativas a la combinación de estos recursos. En (Cooper, 1995) se demuestra que este tipo de problemas es NP-completo. Por ello se presta especialmente a su resolución mediante algoritmos de Simulated Annealing.

En el problema que se presenta a continuación se trata de resolver un problema real con el que se enfrenta una Universidad todos los años en cada una de las convocatorias de exámenes para establecer el calendario apropiado de los mismos.

Una universidad imparte asignaturas de distintas carreras, normalmente unas decenas de ellas. Cada una de las carreras cuenta con un número de asignaturas que puede oscilar entre una veintena y cerca de un centenar. El método tradicional de establecer los calendarios de exámenes suele basarse en utilizar un calendario que existe previamente y que, a veces, no se sabe con qué criterios se generó, sobre el que se realizan leves modificaciones que consisten básicamente en realizar una traslación de fechas. Este sistema tiene las siguientes desventajas:

1) El proceso de asignación de fechas resulta oscuro para los alumnos, generando casos de desconfianza.
2) No hay un criterio unificado para decidir las prioridades en el tiempo de los exámenes de ciertas asignaturas.
3) Suele haber un conjunto importante de estudiantes que muestran su descontento con el calendario y un cierto número de los mismos con solapes.
4) Resulta complicado establecer relaciones entre asignaturas similares de diferentes carreras que permita optimizar los recursos existentes.
5)

Si se quisiera resolver por fuerza bruta un calendario para 50 asignaturas durante un periodo de 20 días posibles de exámenes, con exámenes por la mañana y por la tarde, se dispondrían de unas 10^{80} posibles configuraciones. La exploración exhaustiva de este espacio de soluciones resulta evidentemente intratable.

En el problema se encuentran los siguientes elementos:

1) Un conjunto de carreras para las que hay que generar un calendario de exámenes

2) Un conjunto de asignaturas de las que tendrá exámenes alumnos de cada una de las carreras.
3) Un conjunto de fechas de exámenes en las que se podrán fijar la fecha de realización de los exámenes para cada una de las asignaturas.
4) Un conjunto de restricciones sobre la posible realización del examen de determinadas asignaturas.

En cuanto a las fechas de exámenes se establecen de acuerdo con el calendario de cada una de las convocatorias. Normalmente el número de fechas disponibles suele variar dependiendo de la convocatoria (Febrero, Junio y Septiembre) y la aparición de posibles fiestas, o eventos, entre los días de cada una de las convocatorias.

Las restricciones que se establecen sobre una asignatura suelen ser variadas y desde un punto de vista de asignación de recursos podrían venir condicionadas por el momento de uso de determinadas aulas o la disponibilidad de determinados profesores. Por ejemplo, cierto examen debe realizarse por la tarde, o debe realizarse un viernes que es cuando se puede utilizar determinado laboratorio.

Otro tipo de restricciones vienen condicionadas por el hecho de que dos asignaturas utilicen, por ejemplo, el mismo laboratorio para realizar el examen.

Asimismo, se establece una graduación de las asignaturas para calificarlas como Fácil, Media y Difícil. Esta graduación pretende recoger el hecho de que a una asignatura se le asignará el grado de difícil si el número de alumnos que la repiten es significativamente superior a la media de las asignaturas. De la misma forma a una asignatura se le asignará el grado de Fácil si el número de alumnos que la repiten es significativamente menor a la media. De esta forma y como seguramente haya que celebrar más de un examen el mismo día y hora, se pretende recoger el hecho de que es más difícil que haya posibles solapes para algún alumno en asignaturas con el grado de Fácil que con el grado de Difícil.

4.1. Modelo del problema

4.4.1. Elementos dependientes del problema

Teniendo en cuenta lo descrito se puede formular el problema de la siguiente forma:

Sea $S = \{A, F, C\}$ donde cada uno de los términos se define de la siguiente forma:

1. A es el conjunto de asignaturas de la carrera. Con cada asignatura se recoge el nombre de la asignatura, el curso al que pertenece, el grado y las preferencias de horario de celebración del examen.
2. F es el conjunto de fechas de exámenes, donde cada fecha de examen viene dado por un día y una hora.
3. C es la función de coste, donde se tiene en cuenta el conjunto de restricciones que se establecen sobre el conjunto de asignaturas y sus relaciones temporales.

La estructura del conjunto de soluciones tiene la forma de la Ecuación 4.1:

$$S = \{(a, f) / a \in A, f \in F\} \qquad \text{(Ec. 4.1)}$$

De acuerdo con esta definición |S| = |A|.

Una vez establecido el modelo para el problema hay que establecer cómo ir generando nuevas soluciones en la vecindad una solución dada. Nuestra definición de vecindad viene dada por:

$$S' = \{(a, f') / a \in A, f' \in F\} / |S \cap S'| = |S| - 1 \text{ (Ec. 4.2)}$$

Es decir, se obtiene un vecino a partir de una solución S eligiendo una tupla al azar $(a, f) \in S$ y se cambia su fecha por otra fecha $f' \in F$ para esa asignatura elegida al azar del conjunto de fechas. Si en la tupla elegida la asignatura tiene restricciones en cuanto a la celebración del examen, por ejemplo, tiene que ser un examen de mañana, sólo se seleccionará del conjunto de fechas de examen una fecha que sea de mañana, de manera que el nuevo vecino pertenezca al espacio de soluciones. Con esta definición el tamaño de la vecindad es:

$$|S'| = |A||F| \quad \text{(Ec. 4.3)}$$

Por otra parte, la solución inicial del sistema será una asignación aleatoria de fechas a las asignaturas.

Para la función de coste se ha establecido crear tres niveles de coste dependiendo de la gravedad de incumplimiento de las mismas:

- Nivel fuerte: Cubre un conjunto de reglas que se deben cumplir obligatoriamente como por ejemplo
  - Dos exámenes del mismo curso no deben coincidir el mismo día y hora
  - No deben coincidir dos exámenes difíciles del mismo curso el mismo día

- Nivel medio: Cubre un conjunto de reglas que se intentará de la mejor manera posible que se cumplan siempre, por ejemplo:
  - Los exámenes de un curso deben estar equiespaciados.
  - Dos asignaturas difíciles de cursos sucesivos no coincidan en día y hora.
  - No deben juntarse exámenes difíciles de un mismo curso.
- Nivel débil: cubre un conjunto de reglas que sólo pretenden mejorar la solución en cierto sentido, por ejemplo:
  - No deben coincidir en el día exámenes de cursos separados menos de tres cursos
  - Los exámenes difíciles no deben realizarse en la última semana de exámenes
  - Para los exámenes difíciles debe haber al menos tres días anteriores sin exámenes del mismo curso.
  - Si se dan dos exámenes del cursos contiguos el mismo día es preferible que el de la tarde sea de mayor dificultad que el de la mañana.

4.4.2. Elementos genéricos

Para elegir la temperatura inicial del sistema hay que hacerlo de forma que se permitan un gran número de aceptaciones ya sean de mejora o de no mejora. Para elegir dicha temperatura, se genera un número fijo de vecinos a partir de la solución inicial. A partir de dicho número de vecinos se calcula el coste de todos ellos y se elige una temperatura t_0 que permite aceptar el 90% de los mismos con probabilidad 0.95.

En cuanto a la elección del proceso de enfriamiento, los experimentos realizados nos han llevado a elegir un proceso exponencial con un factor α de 0.98.

Para la condición de parada del bucle interior se ha utilizado una doble condición, que se acepten al menos |A||F|/10 vecinos nuevos o que se hayan probado hasta |A||F| vecinos diferentes.

En la condición de terminación del algoritmo, en el bucle exterior se ha dispuesto que termine cuando se han realizado 5 descensos de temperatura sin encontrar una mejor solución.

4.4.3. Descripción del programa

En la Figura 4.1 se puede observar un ejemplo práctico de propuesta de exámenes para la carrera de Ingeniería en Informática para la convocatoria de Febrero del curso 2004/2005 después de la ejecución del algoritmo. Como se puede observar, posteriormente a la ejecución del algoritmo se permite que se pueda editar el calendario final.

En la Figura 4.1 se han dispuesto para cada día dos bandas de exámenes una que se realiza por la mañana (a las 9:00) y otra banda de exámenes que se realiza por la tarde (a las 17:00). Esto queda representado por las filas de recuadros en blanco debajo de cada día. Así se puede observar cómo el Lunes 7 de Febrero se realizan los exámenes de las asignaturas 3431 y 5430 ambos por la mañana, el primero de ellos se ha calificado como Fácil y el segundo como Difícil. De esta forma se puede ver como la probabilidad de solapa debería de ser muy pequeña, pues por una parte la asignatura 3431 pertenece a tercer curso y es fácil, mientras que la asignatura 5430 pertenece a quinto curso y es difícil, por lo que cuando el alumno llegue a tener el examen de quinto curso es muy probable que ya haya aprobado la asignatura de tercero.

En la misma figura también se pueden observar las restricciones que se han establecido sobre algunas asignaturas. De esta forma se puede ver que la asignatura 1431 con grado de dificultad Medio, programada para el jueves 3 de febrero aparece al final la letra f. Esta letra significa que esta asignatura tiene una asignación fija. Se quiere que se realice ese día y hora. El algoritmo no puede modificar la fecha del examen. Así mismo, se puede observar que la asignatura 1430 con grado de dificultad Difícil, programada el miércoles 16 de febrero, tiene una letra m al final indicando que es una asignatura que sólo se puede programar en un horario de mañana, pero puede ser cualquier día del periodo de exámenes.

Fig. 4.1. Pantalla de presentación de resultados de la ejecución del algoritmo de recocido donde se pueden editar los resultados finales.

Como también se puede ver en la Figura 4.1, se han resaltado los exámenes de primer curso para ver la distribución lograda de los mismos durante el periodo de exámenes. Fíjese que la distribución obtenida para ellos resulta relativamente uniforme. Nótese también que se ha programado que los días 31 de enero y 1 de febrero de 2005 no se realizarán exámenes.

5. Conclusiones

En este artículo se ha tratado de forma general el problema de optimización combinatoria, se ha descrito donde se encuadran este tipo de problemas teniendo en cuenta la teoría de complejidad algorítmica y se ha visto una clasificación general de los distintos tipos de metaheurísticas que se utilizan para abordar este tipo de problemas.

A continuación se ha descrito con cierto detalle la metaheurística Simulated Annealing y sus características generales. En particular nos interesa el problema de convergencia asintótica del algoritmo al conjunto de soluciones óptimas. En este sentido esta metaheurística permite garantizar bajo ciertas condiciones que se puede llegar a obtener la solución óptima al problema.

Para finalizar se ha utilizado Simulated Annealing para el problema de creación de calendarios de examen para la Universidad Alfonso X el Sabio. Del uso del mismo se pueden extraer entre otras las siguientes conclusiones:

- El algoritmo Simulated Annealing resulta relativamente sencillo de implementar y de probar.
- Es necesaria una etapa de afinamiento del algoritmo al problema que se desea resolver, Ello permite utilizar eficazmente los recursos computacionales disponibles.
- El algoritmo se comporta de forma muy robusta frente a cambios en los parámetros del mismo. Cambios en el tamaño de la cadena de Markov, la temperatura inicial, la temperatura final, el número de repeticiones, etc. ha conducido de forma general siempre a "buenas" soluciones, donde lo que cambia es la varianza y valor medio de las soluciones obtenidas, pero no el mejor valor obtenido.
- Resulta un tipo de algoritmos de gran interés y de gran aplicabilidad. En este sentido estamos ya utilizándolo para asignación de profesorado y en el futuro se espera poder abordar problemas de mayor complejidad.

6. Bibliografía

AARTS, Emile H.L., KORST, Jan H. M. y VAN LAARHOVEN, Peter J. M. (1997) *Local Search in Conbinatorial Optimization*. 91-120. John Wiley and Sons.

AARTS, Emile H.L., KORST, Jan H. M. (2002). "Selected Topics in Simulated Annealing", en P. Hansen y C.C. Ribeiro (Eds), *Essays and Surveys on Metaheuristics*. Capítulo 1, Norwell, MA, Kluwer Academic Publishers, pp. 1-37.

ABRAMSON D, KRSHANAMOORTHY M y DANG H. (1996) *Simulated Annealing Cooling Schedules for the School Timetabling Problem*.

COHN, H. y FIELDING, M. (1999). *Simulated Annealing: Searching for an Optimal Temperature Schedule*. SIAM Journal of Operational Research. 135(1), pp 83-96.

COOK, S, (1971). *The Complexity of Theorem Proving Procedures*. Proceedings Third Annual ACM Symposium on Thoery of Computing, pp 151-158.

COOPER, Tim B., KINGSTON, Jeffrey H. (1995). *The Coomplexity of Timetable construction Problems*. Procedings of the First International Conference on the Practice and Theory of Automated Timetabling (ICPTAT'95).

DORIGO M., MANIEZZO V., COLORNI, A. (1996). *Ant System: Optimization by a colony of cooperating agents*. IEEE Transactions on Systems, Man, and Cybernetics-Part B, 26(1):29-41

ELMOHAMED, Saleh M.A. y GEOFFREY Fox, (1998) *A comparison of Annealing Techniques for Academic Course Scheduling*. Practice and Theory of Automated Timetabling II, Selected Papers from the 2nd International Conference, PATAT'97.

GAREY, Michael R., JOHNSON, David S. (1979) *Computers and Intractability: A Guide to the Theory of NP Completeness.* W. H. Freeman and Company, San Francisco.

GLOVER, F (1986): Future paths for integer programming and links to artificial intelligence. *Computers and Operations Research.* 5:533-549.

GLOVER, F, LAGUNA, M. (1997): *Tabu Search*, Kluwer.

GLOVER, F, KOCHENBERGER, G. (eds) (2003). *Handbook of Metaheuristics*. Kluwer Academic Publishers.

HAJEK, B. (1988). "Cooling Schedules for Optimal Annealing". *Mathematics of Operations Research.* vol 13, pp 311-329.

HOLLAND, J. H. (1975). *Adaption in natural and artificial systems*. The University of Michigan Press, Ann Harbor, MI.

JACOBSON, SHELDOM H. (2004). *Analizing the Performance of Generalized Hill Climbing Algorithms*. Journal of Heuristics. Vol 10, pp. 387-405. Kluwer Academic Publishers.

KIRKPATRICK S., GELATT C.D. y VECCHI M.P, (1983) *Optimization by Simulated Annealing*, Science, vol. 220, pp. 45-54

LAGUNA, M., GONZÁLEZ-VELARDE, J.L, (eds) (2000). *Computing Tools for Modeling, Optimization and Simulation*. Kluwer Academic Publishers.

LOZANO, J.A., LARRAÑAGA, P. (2001). *Estimation of Distribution Algorithms. A New Tool for Evolutionary Computation*. Kluwer Academic Publishers.

MARTÍ, R. (2003) Multistart Methods, en Fred Glover y Gary A. Kochenberger (eds), *Handbook of Metaheuristics*, pp 355-368, Kluwer Academic.

MELIÁN, B, MORENO, J,A., MARCOS, J (2003), Metaheuristics: A Global View. *Revista Iberoamericana de Inteligencia Artificial* No. 19, pp. 7-28

MICHALEWICZ, Z., FOGEL, D.B., (2000) *How to Solve It : Modern Heuristics*. Springer Verlag.

MINSKY, Marvin L. (1967) *Computation: Finite and Infinite Machines.* Prentice-Hall Inc. Englewood Cliffs, N.J.

MOSCATO, P, COTTA-PORRAS, C. (2003). Una introducción a los algoritmos meméticos. *Inteligencia Artificial, Revista Iberoamericana de Inteligencia Artificial*. vol 19.

OATES, M.J., CORNE, D.W., SMITH, G.D., (eds) (2000). *Telecommunications Optimization: Heuristics and Adaptive Techniques*. Wiley.

RAJESAKARAN, S., REIF, J.H. (1992). *Nested Annealing: A Provable Iimprovement to Simulated Annealing.* Theoretical Computer Science. (99)-1, pp 157-176.

RIBEIRO, C.C., HANSEN, P. (eds) (2001) *Essays and Surveys in Metaheuristics*. Kluwer.

THEODOSOPOULOS, T.V. (1999). "Some Remarks on the Optimal Level of Randomization in Global Optimization", en P. Pardalos, S. Rajasekaran y J. Rolim (eds). *Randomized Methods in Algorithm Design* en DIMACS Series in Discrete Mathermatics and Theoretical Computer Science. American Mathematical Society, pp. 303-318.