Solución tomada de [1] con fines académicos:

Araujo, Lourdes. Algoritmos Evolutivos: teoría y casos prácticos: Lourdes Araujo, Carlos Cervigón (Spanish Edition) (p. 187). Edición de Kindle.

PLANIFICACIÓN DE HORARIOS

1. DESCRIPCIÓN DEL PROBLEMA

El objetivo de este proyecto es implementar un algoritmo evolutivo para diseñar los horarios de una facultad o escuela. La distribución y coordinación de carga académica en las carreras universitarias es un proceso que involucra numerosos recursos y variables, como por ejemplo los horarios disponibles de los profesores, y la cantidad de aulas y de asignaturas correspondientes a un curso. A veces se encuentran soluciones válidas, aunque no óptimas, y a veces ni siquiera existe una solución que respete todas las restricciones.

Cuando existen pocas variables críticas que intervienen en el problema y una alta disponibilidad de recursos a asignar, es fácil determinar soluciones válidas, e incluso determinar a priori si existe o no un óptimo.

Por el contrario, cuando intervienen una alta cantidad de factores críticos, encontrar una solución ideal es casi imposible, por ello se deben establecer condiciones iniciales que restrinjan el espacio de posibles soluciones pero que permitan encontrar la solución que más se acerca a la óptima.

Los algoritmos tradicionales no aseguran una solución óptima y rápida para la asignación de horarios. Por esta razón la programación evolutiva es una alternativa a considerar.

El algoritmo debe tener en cuenta las siguientes restricciones:

- Dos clases no pueden darse simultáneamente en la misma aula.
- Un profesor no puede impartir clase simultáneamente en dos aulas.
- Cada profesor tiene unos horarios disponibles, y entre estos establece preferencias.

Entradas al programa

- Lista de asignaturas a las que hay que asignar horario:

Código asignatura	Nombre asignatura
101	S1
102	S2
…	…

- Lista de profesores con las asignaturas que imparte cada uno y sus disponibilidades y preferencias de horario.

Por ejemplo, a cada horario se le asigna un coste cuyo valor es 0 si es un horario posible y preferente, y 1 si es posible pero inconveniente.

Cód prof.	Nombre prof.	Cód. asig	Horario	Coste
20	P1	101	H1	0
20	P1	101	H3	1
20	P1	101	H4	0
20	P1	101	H2	1
20	P1	105	H2	0
…	…	…	…	…

- Lista de aulas disponibles para cada uno de los horarios.

Nombre aula	Horario
A1	H1
A1	H2
A1	H3
A2	H2
…	…

Estas entradas se leen de archivos. La salida es una tabla, con una asignación de horario para cada una de las asignaturas. Los resultados, además de salvarse en un archivo, pueden presentarse en pantalla, de forma que el usuario, de forma interactiva, puede consultar el horario asignado a un profesor en una o en todas sus asignaturas, las aulas asignadas a una asignatura, etc.

2. DISEÑO DEL ALGORITMO

Se trata de un problema de minimización, para el que aplicaremos el esquema de algoritmo evolutivo que corresponde. Es también un problema de restricciones que trataremos con la técnica de penalización.

Comencemos por considerar una posible representación de los datos de entrada al problema. Tenemos que representar información sobre aulas, profesores y asignaturas, procurando facilitar el acceso desde cada uno de estos datos a aquellos con los que se relaciona. Comencemos por los profesores. Para cada uno de ellos tenemos que representar un código asignado, su nombre y la lista de asignaturas que se encarga de impartir, que es un dato que no puede variarse. Las asignaturas de un profesor se indican por su código, y para cada una de ellas se tiene una lista de posibles horarios y un coste asignado a cada horario, según las preferencias del profesor.

Podemos representar los costes de los horarios de un profesor para una determinada asignatura mediante la estructura de datos (o clase) *TLCost_Hora_Prof*, que consiste en una lista de posibles horarios para que el profesor imparta la asignatura considerada y su coste asociado.

```
tipo TLCost_Hora_Prof: vector de TCost_Hora_Prof;
tipo TCost_Hora_Prof = registro{
  cadena_caracteres horario; // nombre del horario
  entero coste; // coste del horario
}
```

Representamos ahora una lista de asignaturas, identificadas por sus códigos, a cada una de las que se asocia su lista del tipo *TLCost_Hora_Prof*. Se trata del tipo *TLCost_Asig_Prof*, que es una lista de estructuras *TCost_Asig_Prof*:

```
tipo TCost_Asig_Prof = registro{
   entero codigo_asig; // código de la asignatura
   TLCost_Hora_Prof Horarios_Coste; // lista de posibles
   // horarios con sus costes para la asignatura
}
tipo TLCost_Asig_Prof: vector de TCost_Asig_Prof;
```

Finalmente, podemos representar a un profesor con la estructura *TProfesor*, incluyendo su nombre, su código y su lista de asignaturas asignadas con sus posibles horarios y costes asociados. *TProfesores* representa la lista de todos los profesores.

```
tipo TProfesor = registro{
   cadena_caracteres nombre; // nombre del profesor
   entero codigo; // código del profesor
   TLCost_Asig_Prof conj_asig;
   // asignaturas (con su coste por horario)
   // asignadas al profesor
}

tipo TProfesores: vector de TProfesor;
// lista de profesores
```

La representación de cada aula incluye su nombre y la lista de nombres de horarios en los que está disponible. El conjunto de todas las aulas se representa por la estructura *TLAulas*, que es una lista de estructuras *TAula*.

```
tipo TLHorarios: vector de cadena_caracteres; // lista
de nombres de horarios
// Estructura para representar
// la información sobre una aula

tipo TAula = registro{
   cadena_caracteres nombre; // nombre del aula
   TLHorarios horarios;
// horarios en los que está disponible
}
tipo TLAulas: vector de TAula; // lista de aulas
```

Por su parte la representación de las asignaturas incluye el nombre, el código de la asignatura y el profesor que la imparte. Para hacer más eficientes los accesos a la información sobre el profesor, éste se representa por su posición en la lista de profesores. La estructura para representar la información sobre una asignatura es la siguiente:

```
tipo TAsignatura = registro{
   cadena_caracteres nombre; // nombre de la asignatura
   entero codigo; // código de la asignatura
   entero pos_prof;

// posición (en la lista de profesores)
// del profesor asignado a la asignatura

}
tipo TLAsignaturas: vector de TAsignatura;
// lista de asignaturas
```

2.1 Representación de los individuos

Los individuos pueden representar una asignación de horarios a cada una de las asignaturas a considerar. Por lo tanto, podemos representar a los individuos como listas de genes en las que cada gen representa a una asignatura y el horario asignado a ella. Incluimos también el coste asociado para hacer más eficiente el cálculo de la adaptación:

```
tipo Tgen = registro{
   entero cod_asig; // código de la asignatura
   entero coste; // coste asociado
   cadena_caracteres horario; // nombre del horario
}
```

Luego, cada individuo se representa como una cadena de genes como los descritos, junto con la información usual que necesita un algoritmo evolutivo.

```
tipo TGenes : vector de  TGen;
tipo TIndividuo = registro{
   TGenes genes; // cadena de genes(genotipo)
   real adaptación; // función de evaluación
   real puntuacion;//punt. rel.:adaptación/sumadaptación
   real punt_acu; // puntuación acumulada para sorteos
}
```

2.2 Generación de la población inicial

Los individuos de la población inicial se generan asignando a cada asignatura un horario aleatorio de entre los posibles que tiene disponibles el profesor de la asignatura que la imparte. Un posible esquema es el siguiente:

- Para cada asignatura:
  - Se busca el profesor asignado para impartirla.
  - Se busca la asignatura entre las del profesor.
  - Se elige aleatoriamente uno de los posibles horarios de esa asignatura entre los horarios del profesor.
  - Se asigna el coste que le corresponde.

2.3 Función de adaptación

La función de evaluación debe penalizar cada violación de las restricciones del problema y considerar el coste que supone para cada profesor cada uno de sus horarios disponibles.

Un posible esquema de la función de adaptación es el siguiente:

```
funcion adaptacion(TIndividuo individuo, ...
{
   real adapt;
   entero nph, num_hor;
         ...

   adapt = 0;
   // se suma el coste de los horarios asignados
   // a cada asignatura
  para cada asignatura del individuo hacer
     adapt = adapt + individuo.genes[i].coste;

   // se penalizan las situaciones IMPOSIBLES
   // un profesor no puede estar en dos aulas a la vez

  para cada asignatura del individuo hacer{
     prof = profesor de la asignatura
     nph = num. de apariciones de prof en el mismo horario;
     si nph > 1 entonces
        adapt = adapt + (nph - 1)* PENALIZACION;
  }

  // no puede haber dos asignaturas en la
```

```
    // misma aula y horario
    para cada asignatura del individuo hacer{
       hor = horario de la asignatura
       num_hor = num. de apariciones de hor;
       si num_hor > num_aulas entonces
          adapt = adapt + (num_hor - num_aulas)* PENALIZACION;
    }
    devolver adapt;
 }
```

La adaptación de un individuo, que en este proyecto es un valor a minimizar, recoge la suma de los costes asociados a los horarios asignados a cada asignatura, que se corresponden con las preferencias de los profesores. A este valor se le añade una penalización, dada por el parámetro *PENALIZACION*, por cada situación imposible a que da lugar la asignación de horarios del individuo. En primer lugar se comprueba la existencia de situaciones en que un profesor está a la vez en dos aulas. La variable *nph* lleva cuenta del número de veces que aparece un profesor en el mismo horario. Cada vez que *nph* es mayor que 1, se penaliza la adaptación. Después se comprueba que no haya dos asignaturas en la misma aula y horario. Para hacerlo se cuenta el número de veces que se usa un mismo horario, y si este valor es mayor que el número de aulas, se penaliza la adaptación.

2.4 Operador de cruce

Podemos aplicar un sencillo cruce monopunto que intercambie en los hijos los horarios asignados en los padres antes y después de la asignatura elegida como punto de cruce.

También podemos aplicar un cruce multipunto e intercambiar el segmento de horarios que se encuentre entre dos puntos de cruce elegidos aleatoriamente.

2.5 Operador de mutación

Un sencillo operador de mutación que podemos utilizar consiste en seleccionar aleatoriamente una posición o gen del individuo y cambiar el horario asignado a la asignatura de esa posición. El nuevo horario también debe estar entre los posibles para el profesor que imparte la asignatura.

2.6 Consideraciones adicionales

Para realizar un estudio de los parámetros más adecuados para el algoritmo se puede implementar un sistema que permita variar los parámetros (tamaño de la población, número límite de iteraciones del algoritmo, porcentaje de cruces, porcentaje de mutaciones) interactivamente.

En este proyecto también es interesante variar la penalización asociada a la violación de restricciones. Por ejemplo hay que estudiar qué penalización es necesaria aplicar al caso en que se asigna a un profesor un mismo horario para dos asignaturas para garantizar que no se dé el caso, salvo si el problema no tiene solución.

Hay que tener en cuenta que usar una penalización excesiva para las violaciones de restricciones también puede dar lugar a malos resultados. Si el valor de esta penalización es órdenes de magnitud superior a los costes asociados a las preferencias de los profesores, el mecanismo de selección no tendrá en cuenta dichas preferencias, ya que las diferencias entre unos y otros individuos con una penalización por violación de restricciones serán despreciables.

3. TRATAMIENTO ALTERNATIVO DE LAS RESTRICCIONES

En lugar de considerar las restricciones del problema con técnicas de penalización, pueden probarse otros enfoques, y comparar los resultados obtenidos en ambos casos.

Podemos intentar generar individuos que no violen las restricciones del problema. Por ejemplo, podemos considerar un esquema de generación de individuos de la población inicial como el siguiente:

- Para cada asignatura:
  - Se busca el profesor asignado para impartirla.
  - Se busca la signatura entre las del profesor.
  - Mientras no se asigne horario y quede alguno por probar hacer lo siguiente:

- Se elige aleatoriamente uno de los posibles horarios de esa asignatura y que no se haya probado antes, y se marca como candidato.

- Se comprueba que en ese horario el profesor no tenga otra asignatura. Si es así se pasa a probar otro horario.

- Se comprueba que en ese horario no estén todas las aulas ocupadas. Si es así se pasa a probar otro horario.

- Si se ha conseguido asignar horario se asigna el coste que le corresponde. En otro caso se deshecha el individuo.

Este método tendrá dificultades para encontrar soluciones en problemas muy restringidos.

4. CON OTRAS RESTRICCIONES

Podemos considerar versiones más sofisticadas del problema, que tengan en cuenta otros elementos que también son importantes en la planificación de horarios. Algunas posibilidades son:

- Tener en cuenta el tamaño de las aulas, y el tamaño de los grupos de alumnos de cada asignatura.

- Permitir especificar otras preferencias, aparte de las de los profesores. Por ejemplo, preferencia por que una asignatura que se imparte varios días a la semana lo haga siempre en el mismo horario.

- Control del número máximo de horas diarias de clase para los alumnos y quizás también para los profesores.