Resolución del problema de asignación de fechas de exámenes con evolución gramatical

Alejandro Bellogín Kouki
alejandro.bellogin@uam.es

6 de febrero de 2008

1. Introducción

El problema de encontrar un calendario óptimo de exámenes para unas fechas determinadas se sabe que es NP-completo debido a su gran cantidad de restricciones[2]. Esto es así ya que, en el caso general, hay que controlar que un profesor no tenga dos exámenes a la vez, ni un alumno, que haya aulas disponibles y que la configuración de los exámenes no sea muy desfavorable para los alumnos (no se quieren exámenes próximos que sean del mismo año, obligatorios o duros).

En este trabajo nos centraremos en esta última condición ya que no se dispone del resto de datos. Para resolverlo plantearemos la utilización de la evolución gramatical por medio de una gramática de Christiansen, de manera que no se generen individuos inválidos. Aunque no se dispone de resultados, esperamos obtenerlos en un tiempo no muy largo, y nuestra intención es analizar el control sobre la invalidez de los individuos para el cual se consiguen mejores resultados.

2. Trabajo relacionado

En este trabajo se empleará la evolución gramatical con gramáticas de Christiansen tal y como se explica en [3].

Por otro lado, la función de fitness está basada en la que se encuentra en [1], aunque la hemos adaptado para que no tenga en cuenta los casos que ya contempla la gramática. No obstante, sí utiliza las colisiones entre asignaturas debidas a que pertenecen a cursos consecutivos o al mismo.

3. Propuesta

Ahora presentaremos la gramática de Christiansen que consideramos puede funcionar para resolver este problema. No hemos podido hacer experimentos con ella y, por tanto, no tenemos ningún resultado sobre su validez, pero la hemos diseñado de manera que sea sencillo modificar la cantidad de semántica que es capaz de controlar. Además, también se presenta la función de fitness que evaluaría la bondad de cada solución o individuo.

Antes de comenzar a explicar estas dos componentes vamos a enseñar qué datos de partida necesitan y cómo se obtienen. Como se verá a continuación, la gramática sólo necesita conocer unos identificadores para cada asignatura y un número que identifique cada fecha disponible; para obtener estos datos se realiza lo siguiente:

- Tenemos almacenada en una base de datos todas las carreras de las que se quiere realizar el calendario de exámenes, de esa manera dependiendo de cuál de ellas en particular se quiere saber su calendario, se generará una gramática que incluya los identificadores de las asignaturas que existen en dicha carrera (y para el semestre seleccionado).
- Para generar la lista de fechas disponibles de exámenes se toman como parámetros la fecha inicial, la final y una lista de festivos adicionales; a partir de ellos se genera una lista donde se incluyen por cada día de la semana laborable incluído en el intervalo anterior dos índices: uno indicando que el examen se realizará en horario de mañana y otro en horario de tarde. También se consideran los sábados, pero sólo se permite un examen ese día.

Además, se necesita una tabla de colisiones creada a partir de las asignaturas involucradas
Ahora ya disponemos de los ingredientes para elaborar nuestra gramática de Christiansen y la función de fitness asociada.

3.1. Gramática

Los terminales de nuestra gramática estarán formados por el símbolo igual (=), los identificadores de los eventos (e_i) y los identificadores de los periodos (p_j). Como ya hemos comentados, esta lista se genera dinámicamente en función de los datos de entrada (fecha inicial, carrera elegida, ...).
Los no terminales y sus atributos son los siguientes:

$$S(g_i, g_o), \text{Asignacion}(g_i, g_o), E(g_i, g_o, id), P(d)$$

Donde S es el axioma, E representa un evento, P un periodo y Asignacion relaciona un evento con un periodo. Las reglas de la gramática son:

$$\begin{aligned}
S(g_i, g_o) &\longrightarrow \text{Asignacion}(g_i, g_o) \quad \{AS_0\} \\
S(g_i, g_o) &\longrightarrow \text{Asignacion}(g_i, g_a)\ S(g_a, g_o) \quad \{AS_1\} \\
\text{Asignacion}(g_i, g_o) &\longrightarrow E(g_i, g_o) = P \quad \{AS_2\} \\
E(g_i, g_o, id) &\longrightarrow e_0 | \cdots | e_n \quad \{AS_{e_i}\} \\
P(d) &\longrightarrow p_0 | \cdots | p_m \quad \{AS_{p_j}\}
\end{aligned}$$

Las acciones semánticas indicadas son las siguientes (se necesitan dos atributos globales: el número de asignaturas que se llevan leídos y una tabla con las asignaciones realizadas):

$$\begin{aligned}
AS_0 &\equiv \text{if(numAsig} + 1 != |E|)\ \text{error();} \\
AS_1 &\equiv \text{numAsig} + + \\
AS_2 &\equiv \text{if(existConflict}(E.id, P.d)) : \text{error(); else : table.put}(E.id, P.d); \\
AS_{e_i} &\equiv g_o = g_i - \{E \rightarrow e_i\}; E.id = e_i; \\
AS_{p_j} &\equiv P.d = p_j;
\end{aligned}$$

Todas las acciones semánticas son necesarias:

- Utilizamos AS_0 para comprobar que se ha generado un individuo que asigna una fecha a **todas** las asignaturas.
- AS_1 se utiliza para actualizar el contador de asignaturas leídas.
- AS_2 es la acción semántica más importante: comprueba si hay conflicto entre la asignatura que se acaba de asignar y las ya asignadas. El método `existConflict` nos permitirá modificar la cantidad de semántica que será capaz de añadir la gramática, ya que podremos modificar dicho método para que devuelva verdadero en distintas situaciones:

- Si hay asignaturas asignadas del mismo curso a la misma hora.
- Si hay asignaturas asignadas del mismo curso el mismo día.
- Si se han asignado asignaturas en cursos adyacentes el mismo día (o incluso a la misma hora).
- ... (otras variantes: restricciones más finas)

En el caso en que no haya conflicto, aceptamos la asignación como válida y la guardamos en la variable global `table`.

- En las acciones semánticas relacionadas con los eventos obtenemos el id del evento y nos aseguramos que no se vuelve a generar, modificando la gramática.
- Las acciones semánticas relativas a los periodos simplemente obtienen el identificador de dicho periodo.

3.2. Función de fitness

La función de fitness evaluará el atributo `table` generado por la gramática utilizando la matriz de colisiones creada al comienzo de la ejecución. Podrá tener en cuenta la dificultad de las asignaturas, los días de separación entre ellas y los cursos a los que pertenecen, dando un valor bajo a los elementos mejores (debido a que suma constantes dependiendo de la colisión encontrada); como queremos maximizar una función en lugar de minimizarla, a este valor calculado se le aplica la función $f(x) = \frac{1}{x+1}$.

Hay que notar que la función de fitness no deberá hacerse cargo de controlar que un individuo haya asignado una fecha a cada una de las asignaturas, ya que esto nos lo asegura la gramática, al igual que el que se haya asignado más de una fecha a una misma asignatura; por el contrario, deberá evaluar las distintas colisiones entre las asignaturas en función de lo que sea capaz de validar la gramática (cuanta más semántica se añada, menos trabajo tendrá que realizar la función de fitness, pero tendremos algo más parecido a una búsqueda ciega en vez de a una búsqueda dirigida).

En el siguiente pseudocódigo podemos ver la función de fitness:

```
fitness = 0;
for i:=0 to max do
begin
        for j:=0 to max do
        begin
                if i != j AND collision[i][j] > THRESHOLD_COLL then
                        factor = getDifference(i, j, table);
                        fitness = fitness + factor * collision[i][j];
                else
        end
end;
```

en esta función se utiliza la tabla generada por la gramática (`table`), la matriz de colisiones, una constante de umbral a partir del cual se consideran las colisiones (si queremos considerar colisiones más o menos graves) y que deberá utilizarse también en la función `existConflict` utilizada en la gramática. En la función de fitness se llama a otra función, `getDifference`, que será la que nos diga si la distancia a la que están dos asignaturas asignadas (mirando la tabla) es relevante o no (por ejemplo, a partir de una distancia determinada, el número devuelto será cero, queriendo esto decir que esas asignaturas no interfieren).

4. Trabajo futuro y conclusiones

Se tiene previsto continuar con este trabajo para comprobar la efectividad de la gramática propuesta, así como de los distintos niveles de semántica que se pueden añadir sin obtener resultados muy desfavorables. Para ello se utilizará una herramienta que permite hacer evolución gramatical dada una gramática y una función de fitness.

Referencias

[1] Edmund K. Burke and J. P. Newall. A multistage evolutionary algorithm for the timetable problem. *IEEE Transactions on Evolutionary Computation*, 3(1):63–74, 1999.

[2] Tim B. Cooper and Jeffrey H. Kingston. The complexity of timetable construction problems. In *Proceedings of the First International Conference on the Practice and Theory of Automated Timetabling (ICPTAT '95)*, pages 511–522, 1995.

[3] A. Ortega, de La, and M. Alfonseca. Christiansen grammar evolution: Grammatical evolution with semantics. *Evolutionary Computation, IEEE Transactions on*, 11(1):77–90, 2007.