Planificador de horarios de hostelería

Federico García Ávila

Daniel Corrales Rodríguez

Índice

- Introducción..... *Pág. 3*
  - Especificación del problema.....*Pág. 3*
  - Motivación.....*Pág. 4*
  - Estado del arte.....*Pág. 4*
  - Metodología.....*Pág. 5*
  - Técnicas utilizadas.....*Pág. 5*
- Conclusiones.....*Pág. 7*
- Recomendaciones al ejecutar.....*Pág. 8*
- Bibliografía.....*Pág. 9*

Introducción

Especificación del problema

Para este proyecto se ha desarrollado un planificador de horarios para el personal de hostelería, en concreto de un Rodilla, de forma que mediante nuestra aplicación los responsables de calcular los horarios se librarán de esta tarea.

Este proyecto se ha realizado por etapas:

En la primera etapa se ha realizado una aplicación sin interfaz que crea una tabla de horarios para una semana teniendo en cuenta el número de turnos al día y el número de trabajadores necesarios por turno. Para ello, se tienen en cuenta las restricciones impuestas por el convenio de Hostelería, que son:

- Jornada laboral de 40 horas semanales (8 horas diarias x 5 días de trabajo), sin exceder de ocho horas la jornada laboral diaria. Lo que significa esto para nuestro proyecto es que cada empleado solo puede trabajar un turno al día y suponemos que cada turno será de 8 horas.
- Transcurso de 12 horas entre la finalización de una jornada y el inicio de la jornada siguiente. Para esto se ha añadido una mejora que permite comprobar que un empleado no pueda trabajar el último turno de un día y el primero del siguiente, ya que según el horario del local no pasan 12 horas entre ambos turnos.
- Los trabajadores disfrutan cada semana de dos días de descanso ininterrumpidos, aunque no tienen por qué pertenecer a la misma semana estrictamente (pueden ser lunes y domingo). En nuestra primera versión, por simplicidad, no se permitió que esos días ininterrumpidos sean lunes y domingo. Esto se implementó en la segunda etapa.

La segunda etapa consistió en lo que se ha explicado en el punto 2 y 3 de las viñetas. Para esto se ha incluido una base del conocimiento que nos permite implementar estas funcionalidades. En esta base del conocimiento se guarda el último turno de la semana anterior.

La tercera etapa consistió en añadir un sistema de guardado en fichero para mantener los datos de la base del conocimiento entre ejecuciones (fichero "BDEjecucion.txt").

Como cuarta etapa se ha implementado un sistema de prioridades para los empleados, de forma que cada empleado puede pedir su preferencia de turnos y a lo largo de las semanas se beneficia a todos los empleados por igual. Para conseguir esto, llevamos un sistema de puntos, de forma que cuantos más puntos tiene un empleado, significa que más se le ha favorecido, y por tanto en la duda de elegir entre dos peticiones, se elegirá por medio de los puntos. Estos puntos forman parte de la base del

conocimiento, y se guardan en otro fichero (fichero "BDPuntos.txt"), para que no se pierdan entre ejecuciones.

Como última mejora se le ha añadido una interfaz gráfica en Java para hacer la aplicación más usable.

Motivación

Debido a que conocemos varias personas que trabajan en el mundo de la hostelería nos dimos cuenta de lo ardua que era la tarea de planificar los horarios de un establecimiento cumpliendo con la normativa del estado y atendiendo a las preferencias de cada uno de sus trabajadores a la hora de asignar turnos.

La asignatura de Programación declarativa avanzada nos exigía la realización de un proyecto para poner en práctica las herramientas de programación centradas en resolutores, como el basado en dominios finitos, las cuales nos brindaban la opción de crear un programa capaz de simplificar dicha tarea de planificación.

Aplicar estos métodos, aprendidos en clase, sobre problemas reales que podían tener por ejemplo nuestros conocidos del ámbito de la hostelería e investigar como presentar dichas soluciones mediante una interfaz usable y familiar para el usuario han sido principalmente las motivaciones a la hora de decantarnos por este proyecto.

Pese a que no fuese una motivación inicial, también nos ha servido para aprender mucho más sobre la asignatura, ya que nos hemos tenido que enfrentar a problemas no vistos en clase y buscar soluciones por nuestra cuenta.

Estado del Arte

Tras investigar el estado del arte referente a nuestro proyecto hemos encontrado un par de páginas que nos muestran un "top" de programas de scheduling que podemos encontrar ahora mismo en el mercado:

TopTenReviews [1]:

1. Chaos Intellect
2. VIP Task Manager
3. WorkEngine

Top Scheduling Software Products [2]:

1. Front Desk
2. Web Schedule
3. Aladtec

La mayoría de los programas de sheduling que hemos encontrado son muchísimo más extensos y funcionales que el nuestro, permitiendo un sistema de login a una base de datos, y ofreciendo funcionalidades extra como el scheduling de tareas, sistemas de correo electrónico, sistema de importación de horarios, sistema de encriptación de datos, etc. No obstante,

es necesario mencionar que estos programas son fruto del trabajo de muchas personas en un proyecto a lo largo de mucho tiempo.

Metodología

Para el proyecto se decidió que debido a las características del mismo, la metodología que más se adaptaba a este proyecto era metodología ágil.

"*El desarrollo ágil de software refiere a métodos de ingeniería del software basados en el desarrollo iterativo e incremental, donde los requisitos y soluciones evolucionan mediante la colaboración de grupos auto organizados y multidisciplinarios.*" [3]

Para nuestro caso, la colaboración de grupos auto organizados y multidisciplinarios no tiene mucho sentido dado que solo somos 2 personas trabajando, pero el hecho de realizar las distintas etapas de forma iterativa e incremental si se adapta a como hemos enfocado el desarrollo del proyecto.

Generalmente el trabajo de implementación se ha realizado de forma conjunta exceptuando en algunas tareas, que por su dificultad más leve, se ha decidido dividir el trabajo, aunque siempre manteniendo una comunicación constante entre las 2 partes del equipo.

Técnicas utilizadas

Como está explicado en la introducción, primero empezamos con la funcionalidad básica, que está únicamente programada en el lenguaje Prolog, mediante el uso de los Dominios Finitos implementados mediante el modulo "clpfd" [4].

Antes de seguir, y para poder explicarnos claramente, hay que mencionar que en la aplicación los datos que se piden son: el número de turnos por día (que llamaremos *nTurn*) y el número de personas que trabajan en cada turno (que llamaremos *nPers*). Una vez que sabemos estos datos, podemos calcular mediante una fórmula simple el número de trabajadores totales del establecimiento (que llamaremos *numT*).

La funcionalidad básica al principio nos causó grandes problemas, ya que el tiempo de ejecución se disparaba a partir de $numT = 3$ (con $numT = 6$ ya no se podía llegar a ver la solución, porque tardaba demasiado en terminar). Esto estaba causado por meter tantas restricciones de dominio finito que dependían de otras, que pedían mucho trabajo al resolutor de Prolog, junto con la forma de trabajar de Prolog de dar varias soluciones a funciones sencillas.

Mediante un análisis de la ejecución del programa, nos dimos cuenta que mediante la inserción de cortes (!) en puntos clave del código, se conseguía reducir el tiempo enormemente. Además, descubrimos y arreglamos un error por el que, por alguna razón, al crear las listas sobre las que luego se aplicaban dominios finitos, las primeras listas que creaban eran correctas, pero al darle a ";", el programa se quedaba colgado esperando la siguiente solución. Una vez arreglado esto, conseguimos que el tiempo de ejecución con el *numT* = 3 fuese menos de un segundo.

La ejecución con numT = 6 (nPers = 2, nTurn = 2), es también bastante rápida, siempre que en la semana anterior, el domingo haya dos personas librando o que en general la semana tenga unas pocas restricciones.

También usamos la capacidad de Prolog de tener una base del conocimiento de forma sencilla, mediante el uso del "assert"[5] y "retractall"[5]. A esto le añadimos la capacidad de uso de ficheros[6].

Por último, usamos la librería "jpl"[7], que se encuentra en la carpeta "lib" al instalar el SWI-Prolog, para meterle una interfaz en Java, de forma que la aplicación es mucho más sencilla de utilizar para el usuario medio. Esto nos dio también bastantes problemas, ya que la documentación de esta librería no es muy buena y además tiene varios bugs, sin solución, por los cuales hemos tenido que adaptar la estructura de la aplicación y ligeramente la funcionalidad.

Conclusiones

La combinación de Prolog con dominios finitos permite solucionar este problema de manera más o menos sencilla, ya que solo hay que poner las restricciones deseadas, y el propio resolutor se encarga de generar un árbol de búsqueda y buscar valores que cumplan todos las restricciones. A esto, se le añade la sencilla forma de Prolog de usar una base del conocimiento y de gestión de ficheros, de forma que guardar los valores necesarios entre ejecuciones es muy simple.

Tratar de afrontar este tipo de problemas con lenguajes orientados a objetos, como Java o C++, sería una ardua tarea comparado con la menor dificultad y mayor claridad de programación que encontramos al implementarlo con Prolog. Aunque usar Prolog no significa no poder usar Java para crear la interfaz y la gestión de llamadas al resolutor de Prolog, creando una aplicación tan usable como las de los lenguajes de programación más modernos, mientras que su núcleo está implementado con la sencillez que nos proporciona Prolog.

Recomendaciones al ejecutar

Recomendamos usar los siguientes valores:

- nTurn = 2, nPers = 1 -> numT = 3.

  No es necesario meter restricciones para ver la ejecución rápida.

- nTurn = 2, nPers = 2 -> numT = 6.

  La primera ejecución siempre es rápida, ya que no hay que comprobar las restricciones con la semana anterior. Sin embargo, si queremos que la segunda ejecución sea rápida, recomendamos meter restricciones. Lo mejor es meter restricciones con días libres, ya que son las que más agilizan el proceso (con meter dos, debería ser suficiente para que se ejecute bastante rápido).

- No se recomienda meter números más grandes, porque, aunque es posible, esto llevaría a que el programa tardase mucho tiempo en ejecutar, aunque si metemos números más grandes y muchas restricciones válidas (que no se contradigan entre ellas), tardará bastante menos en ejecutar.

Hay que recalcar que para calcular numT, se usa la formula: nTurn*nPers*7/5, lo cual causa que el numero sea decimal, con lo cual cogemos el primer número entero por encima. Esto significa, que normalmente habrá algún día en el que haya más personas trabajando en un turno que las que se necesitan.

Importante: hay que cambiar las direcciones de los ficheros de base de datos en el código Prolog, ya que en cada ordenador serán diferentes. Además, hay que cambiar la librería jpl, ya que cada ordenador puede tener instalado el SWI-Prolog con rutas diferentes. También hay que cambiar la variable Path para que tenga guardadas las carpetas "lib" y "bin" del SWI-Prolog.

Bibliografía

[1] *http://scheduling-software-review.toptenreviews.com/*

[2] *http://www.capterra.com/scheduling-software/*

[3] *http://es.wikipedia.org/wiki/Desarrollo_%C3%A1gil_de_software*

[4] *http://www.swi-prolog.org/man/clpfd.html*

[5] *http://www.swi-prolog.org/pldoc/man?section=db*

[6] *http://www.swi-prolog.org/pldoc/man?section=edinburghIO*

[7] *http://www.swi-prolog.org/packages/jpl/*