Desarrollo de un servicio web de apoyo al proceso de organización de torneos de fútbol mediante programación por restricciones

James Andrés Payán Caicedo

(Proyecto de trabajo de grado)

Escuela de Ingeniería de Sistemas y Computación
Universidad del Valle

Supervisado por: Prof. Juan Francisco Díaz

Noviembre de 2017

Índice general

Prologo

En el presente documento se expone el proyecto COPISTA (COnstraint ProgrammIng Scheduling Tournament footbAll), mediante el cual se desarrolló un servicio web para apoyar el proceso de organización de torneos de fútbol, usando programación por restricciones.

Parte I

INTRODUCCIÓN

Capítulo 1

PRESENTACIÓN DEL PROBLEMA

1.1. Definición del Problema

Ausencia de un servicio web que se integre con el Sistema de Información Nacional del Deporte (SIND), para apoyar el proceso de organización de torneos de fútbol mediante el uso de programación por restricciones.

1.2. Descripción del Problema

El grupo de investigación AVISPA, en nombre de la Universidad del Valle, se encuentra colaborando con el desarrollo del proyecto SIND, sistema de información web interactivo, con el cual se pretende dar a conocer la realidad de los escenarios deportivos, de los grupos humanos, sus preferencias, comportamientos y tendencias, entre otros.

Entre las diferentes necesidades que se ha planteado cubrir con el proyecto SIND, se encuentra la organización de torneos de fútbol a nivel local, donde los principales usuarios son: clubes, ligas, cajas de compensación y demás actores en el ámbito regional.

Sin devaluar el esfuerzo que se está realizando, el grupo de investigación AVISPA es consciente que el desarrollo de un sistema de información, el cual únicamente almacena y centraliza, no es suficiente. Actualmente el SIND carece de un servicio que ofrezca soporte a la realización de las siguientes tareas dentro del proceso de organización de torneos de fútbol:

1. Sortear de forma adecuada los enfrentamientos en la primer ronda de cada torneo, teniendo en cuenta los equipos que se encuentran participando.

2. Programar las fechas y horarios en que se jugarán los partidos de cada torneo, teniendo en cuenta la fecha de inicio y fin del torneo, cantidad de partidos que se debe/puede jugar por semana, la disponibilidad de los escenarios y las recomendaciones de mejores fechas y horarios (tarde o noche) para realizar los partidos según el departamento de mercadeo.

3. Seleccionar los escenarios donde se ejecutará cada partido del torneo de acuerdo a la disponibilidad de horarios e instalaciones. Esta selección de escenarios se debe hacer teniendo en cuenta que cada enfrentamiento corresponde a dos partidos, donde cada equipo tiene la oportunidad de jugar al menos una vez como equipo local.
4. Seleccionar los árbitros que juzgarán cada partido teniendo en cuenta las características, la experiencia, la relevancia del partido para el torneo y la disponibilidad de los jueces.

Este problema afecta de forma directa a federaciones, ligas, clubes, cajas de compensación y centros recreativos.

Capítulo 2

OBJETIVOS

2.1. Objetivo general

Desarrollar un servicio web que sirva de soporte para una organización eficiente, dentro de un intervalo de tiempo factible, de torneos de fútbol en el territorio nacional colombiano, el cual a su vez pueda ser usado a través del Sistema de Información Nacional del Deporte -SIND.

2.2. Objetivos Específicos

El servicio web creado bajo este proyecto debe mitigar el problema identificado mediante el uso del paradigma de programación por restricciones y cubriendo a su vez de forma específica las siguientes tareas:

- Sortear de forma adecuada los enfrentamientos de equipos en la primer ronda de cada torneo.
- Programar las fechas y horarios en que se jugará cada partido de cada torneo.
- Seleccionar los escenarios donde se ejecutará cada partido de cada torneo.
- Seleccionar los árbitros que juzgarán cada partido.

Capítulo 3

ALCANCE DE LA PROPUESTA

El resultado de este proyecto no pretende ser un sistema que aborde todas las tareas necesarias para la organización de torneos de fútbol[1], pero utilizado en conjunto con el SIND y el resto de estrategias que se tenga implementadas dentro de cada federación, liga, club, caja de compensación o centro recreativo, permitirá ofrecer un servicio desde la página del SIND que apoye la realización de dichos torneos.

Específicamente, en este desarrollo no se hará:

- Registro de equipos que participarán en el torneo de fútbol.
- Administración y seguimiento de la ejecución del torneo de fútbol.
- Interacción directa con la base de datos del SIND.

El sistema desarrollado bajo este trabajo de grado se nutrirá con la información entregada como archivo de entrada y como resultado entregará un archivo de salida con la información de la organización del torneo. El formato de dichos archivos y la información necesaria que ellos contendrán, se definirán en la fase de toma de requerimientos.

[1] El proceso de organización y gestión de torneos de fútbol implica diferentes actividades que van desde la recolección de información hasta el registro de ejecución y seguimiento de los mismos.

Capítulo 4

JUSTIFICACIÓN

La organización de torneos de fútbol es una actividad importante para las federaciones, ligas, clubes, cajas de compensación y centros recreativos, puesto que el fútbol es uno de los deportes que despierta mayor interés dentro en la población colombiana[1]. Debido a esto, existe como requerimiento que entre los servicios que ofrece el SIND, haya uno que cubra esta actividad. Este proyecto nace con la esperanza de dar solución en parte a este requerimiento y dejando a disposición un producto que a futuro pueda ser integrado con el proyecto SIND.

Se propone un servicio web debido a que: 1) de esta forma se puede desarrollar el proyecto de forma independiente y paralela al desarrollo del proyecto SIND y 2) favorece la oportunidad de que este producto sea reutilizado en un futuros desarrollos (esto gracias al bajo acoplamiento desde este proyecto hacia los componentes desarrollados en el proyecto SIND).

Teniendo en cuenta la descripción del problema, expuesta en la sección 1.2, la organización de torneos de fútbol apoyado por computadora se puede realizar con diferentes técnicas de programación, tales como: programación dinámica, programación entera y programación por restricciones. Se elige el enfoque de programación por restricciones debido a la facilidad con que se puede describir, mantener y extender el modelo del sistema, lo cual es importante si se tiene en cuenta que los torneos de fútbol pueden diferir en muchos aspectos tanto entre diferentes localidades como a medida que pasa el tiempo, seguramente no son las mismas reglas de organización de torneos hoy que hace 20 años.

Adicionalmente, la organización de torneos es básicamente un problema de scheduling[2] y tal como se puede evidenciar entre las conclusiones de [2], la programación por restricciones se presenta como una alternativa que se adapta de forma natural a problemas de este tipo, brindando no solamente facilidad en la tarea de programación, sino que, comparado con enfoques como programación entera, es más eficiente hallando soluciones.

Plantear el sistema como un servicio web y con un enfoque de programación por restricciones, también aporta a alcanzar dos de los objetivos específicos del proyecto SIND registrado en Colciencias [3]:

[1] El fútbol es uno de los deportes más populares en Colombia, tal como se evidencia en uno de los artículos publicados en la revista Semana [1].

[2] Del inglés, organización de cronogramas.

Realizar el diseño arquitectural que satisfaga las necesidades del Sistema de Información del Deporte con énfasis en su desempeño, escalabilidad y rendimiento.

Crear un sistema que permita capturar y generar información clave para la identificación y caracterización de los actores (individuales, institucionales, corporativos), que incluya elementos de georreferenciación, normatividad vigente y algunos aspectos de tecnologías de big data, analíticas web y programación por restricciones.

Desde el punto de vista académico, este proyecto representa la oportunidad de aplicar los conocimientos adquiridos a través de los diferentes cursos de la carrera de Ingeniería de Sistemas de la Universidad del Valle. Concretamente, se aplicará los conocimientos adquiridos en los cursos: Programación por restricciones, Complejidad y Optimización, Desarrollo web y redes inalámbricas y Desarrollo de software I y II.

Capítulo 5

MARCO TEÓRICO

5.1. Antecedentes

Actualmente el grupo de investigación AVISPA junto con otros entes de la Universidad del Valle, tales como el grupo de investigación GRINDER y el grupo de estudios doctorales GEDI, se encuentran colaborando con el desarrollo del sistema de información SIND. De acuerdo al proyecto registrado en Colciencias [3], el objetivo general de este sistema es:

> Diseñar e implementar el Prototipo Funcional del Sistema de Información del Deporte que permita incluir y caracterizar a los actores del Sistema Nacional del Deporte, ofrecer bienes y servicios del sector para los ciudadanos, que adicionalmente sea una herramienta para generar información de apoyo a la toma de decisiones de política pública para el estado y la sociedad civil.

De acuerdo a lo que manifiestan mencionados grupos de investigación y de acuerdo al compromiso de «ofrecer bienes y servicios», según los objetivos, el desarrollo actualmente carece de un servicio que junto con la información que se está recopilando en el sistema de información, ofrezca un servicio que permita a los entes correspondientes la organización de torneos de fútbol.

Hay una gran cantidad de antecedentes a este proyecto en lo que respecta a la organización de torneos deportivos [4]. En estos se han propuesto diferentes enfoques para la solución de los problemas de organización de torneos: estrategias de descomposición, programación entera, programación por restricciones, búsquedas heurísticas, metaheurísticas y sus híbridos.

La siguiente lista muestra algunos ejemplos de las aplicación de estos métodos para la solución de problemas reales en la organización de torneos de fútbol:

- Organización de torneos de fútbol para niños en Michigan, Estados Unidos, utilizando programación por restricciones [5].
- La aplicación de heurísticas y algoritmos branch and bound en la organización de torneos de las ligas de Austria y Alemania [6].

- La organización de los torneos de la liga mayor de fútbol italiana bajo el enfoque de la programación entera lineal [1] [7].

- Organización de torneos de la liga de fútbol chilena usando programación entera [8].

- El uso de una estrategia de descomposición de programación entera para la organización del torneo de fútbol brasileño [9].

Una rápida búsqueda, en cualquiera de los motores de búsqueda disponibles en internet, nos permite identificar que es amplia la gama de software para organización de fechas y horarios de diferentes eventos, incluyendo torneos deportivos. Sin embargo, como se puede observar en cada uno de los ejemplos anteriores, los requerimientos en cada aplicación suelen ser muy específicos, difiriendo uno de otros, por lo cual hace difícil, sino imposible, la adaptación de estos software a los problemas particulares de cada entidad deportiva en los diferentes países del mundo. El caso de estudio propio no es la excepción, los requerimientos para la solución del problema van desde particularidades en las restricciones para la organización de los torneos, hasta la necesidad de que la herramienta implementada se integre fácilmente con el SIND. Sumado a que en muchos de los casos el precio por la adquisición e implementación de estos software están por fuera del presupuesto de los entes locales[2], hacen de este proyecto una gran oportunidad de negocio.

Finalmente cabe mencionar que dentro del ámbito local, en la Universidad del Valle, bajo la dirección del grupo de investigación AVISPA, se encuentran varios trabajos alrededor de la programación por restricciones. Entre estos se encuentra implementado una aplicación llamada SABIO, la cual permite a los usuarios realizar inferencias sobres los resultados de un torneo fútbol, dado unos datos de entrada previos[10].

5.2. Marco Conceptual

5.2.1. Programación por restricciones

De acuerdo con [11], la programación por restricciones es el estudio de sistemas computacionales que ayudan a resolver problemas por medio de la formulación de restricciones y encontrando las soluciones que satisfagan todas las restricciones declaradas.

Una restricción puede ser vista como la formalización de relaciones del mundo real en forma de abstracciones matemáticas. Dichas abstracciones, básicamente, son relaciones lógicas entre variables desconocidas cuyos valores pueden ser tomados de un dominio dado. Entonces las restricciones restringen los posibles valores de dichas variables. Un aspecto importante es que las restricciones especifican la relaciones que se deben mantener sin declarar los métodos computacionales que las hacen cumplir.

De acuerdo con [12], una vez que se tiene las restricciones, las variables y sus dominios, se puede decir que tenemos un modelo CSP y por ende un espacio de búsqueda en el cual se puede encontrar una solución para nuestro modelo. Después de esto se somete el espacio de búsqueda B al siguiente procedimiento:

[1] En inglés se suele usar ILP-Based approach para referirce al enfoque de programación entera lineal

[2] Refiriéndose aquí nuevamente a las federaciones, ligas, clubes, cajas de compensación y centros recreativos que componen el Sistema Nacional del Deporte.

```
procedimiento p(B):
    Propagar (B)
    si B es estado final:
        retornar solucion B
    si no:
        (B_1, B_2) = Distribuir(B)
        aplicar procedimiento p(B_1)
        aplicar procedimiento p(B_2)
```

5.2.2. Propagación

El proceso de propagación de un espacio de búsqueda, consiste en transformar este espacio en un modelo CSP equivalente. En este nuevo modelo se habrá eliminado de los dominios de las variables aquellos valores que se lograron identificar no hacen parte de la solución [12]. La propagación es uno de los conceptos fundamentales de la programación por restricciones. El inconveniente se encuentra en que para la mayoría de los problemas no es posible descartar todos los valores que no pertenecen a una solución solo con hacer propagación, de ahí que sea necesario hacer distribución.

5.2.3. Distribución

El proceso de distribución de variables en programación por restricciones, hace referencia al proceso de dividir un espacio de búsqueda en dos (o más), de tal forma que la unión de estos nuevos espacios sea equivalente al espacio de búsqueda original.

La distribución se puede lograr mediante diferentes reglas de selección de variables y división de sus dominios, obteniendo así nuevos espacios que tienen dominios de búsqueda más pequeños, garantizando de este modo la completitud del procedimiento p aplicado a los espacios de búsqueda.

Para ampliar la información acerca de las diferentes técnicas aplicadas en el proceso de distribución, puede consultar sobre split en [12] o sobre branching en [13].

5.2.4. Espacios de Búsqueda

En este trabajo se usará el concepto de espacio de búsqueda cómo la instanciación de modelo CSP en el tiempo de ejeción del programa, es decir, un espacio en memoria donde residen las restricciones, las variables y sus dominios.

5.2.5. Branch&Bound, Búsqueda de la mejor solución

El algoritmo Branch&Bound es uno de los algoritmos más usados durante el proceso de distribución cuando se necesita encontrar la mejor solución basado en una función de costo, o función objetivo.

Cada vez que se encuentra una solución en un espacio de búsqueda, este algoritmo guarda el valor de la función objetivo de ese espacio y agrega la restricción de que el valor de la función objetivo, en el resto de espacios de búsqueda que no han sido explorados, sea mejor que el valor encontrado. De esta manera se evita explorar espacios de búsqueda cuyos dominios en las variables no permiten encontrar una mejor solución, así como también se garantiza que cada nueva solución encontrada, siempre va a ser mejor que la anterior. Veáse [12] y [13].

5.2.6. Reducción de problemas

La reducción de un problema A en términos de un problema B es un proceso mediante el cual se transforma el problema[3] A de tal forma que el resultado de la transformación es un problema B. Se dice que el problema A es reducible en términos de B, cuando las soluciones del problema B también son soluciones del problema A. Está técnica suele ser útil en teoría de la computación para hacer demostraciones de complejidad de los problemas [14]. En este trabajo utilizaremos esta técnica para acelerar el tiempo de desarrollo, puesto que en algunos casos ya se encuentra implementada la forma de solucionar problemas que son equivalentes a los problemas de satisfacción de restricciones estudiados en este trabajo.

5.2.7. Servicio web

Un servicio web es un servicio que es ofrecido a través de internet por aplicaciones software para ser utilizado por otras aplicaciones software. Típicamente esta comunicación se realiza transfiriendo entre las aplicaciones archivos de textos en formatos como XML o JSON. La W3C define un servicio web como un sistema software diseñado para soportar la interacción interoperable de máquina a máquina sobre una red de computadores[15].

5.2.8. RESTful API

REST[4], se puede ver como un modelo estándar para construir servicios web [16]. Un servicio web que cumpla con todas las especificaciones del modelo REST se define como un RESTful API [17].

5.2.9. Standard Input/Output

El Standard Input/Output se refiere a los flujos[5] de datos estándar para comunicarse de forma interactiva con los procesos del sistema operativo. Los flujos de datos típicamente son: el flujo de datos de entrada (input), para enviar mensajes al proceso y el flujo de datos de salida (output), para visualizar las respuestas que genera el proceso [18].

[3] En nuestro caso particular, problemas de satisfacción de restricciones
[4] de las siglas en inglés para Representational State Transfer
[5] en inglés stream

Parte II

SISTEMA COPISTA

Capítulo 6

ANÁLISIS CONCEPTUAL DEL PROBLEMA

El diagrama de clases de la figura 6.1 pretende mostrar cómo se relacionan los conceptos en un problema de organización de torneos.

Un torneo tiene una localidad en la que se organiza y una fecha de inicio y de finalización. Además en el caso que se quisiera organizar más de un torneo, estos tienen un nivel de prioridad. En la descripción de los torneos, también se debe indicar cuántos partidos se puede jugar como máximo por día y por semana.

La descripción del torneo también debe indicar los escenarios en los que se pueden jugar los partidos. De estos escenarios se debe saber su nombre, si son iluminados (para luego determinar si se puede jugar de noche) y sus restricciones de uso en el tiempo, es decir, en qué fecha e intervalo de hora no se pueden jugar partidos en estos escenarios.

Para organizar el torneo se hace necesario saber cuáles son los equipos que participan en él. De cada equipo se desea saber su nombre y a que localidad pertenece. Adicionalmente, en el proceso de organización de torneos, se debe indicar cuál será el orden de los equipos, es decir, quién juega primero, segundo, tercero, etc.

El conjunto de jueces que pueden juzgar los partidos también se debe conocer con anticipación. De estos jueces se sabe su nombre, la experiencia y las restricciones de uso, es decir, fechas en que no pueden asistir a un partido.

A los torneos se pueden indicar qué días de la semana, fechas y horarios (tarde o noche) es mejor jugar los partidos para atraer más público.

Finalmente, el objetivo del proceso de organización de torneos es indicar cuáles partidos son necesarios para ejecutar el torneo, señalando las rondas y fechas de cada partido. En el proceso a estos partidos se le asignará un juez, un escenario de juego, los equipos que se enfrentan (para todos los partidos en los torneos todos contra todos o solo en la primera ronda en los torneos por eliminación), el día calendario y la hora de juego.

En los capítulos siguientes se describirá cómo estos conceptos del mundo real, y sus relaciones, se

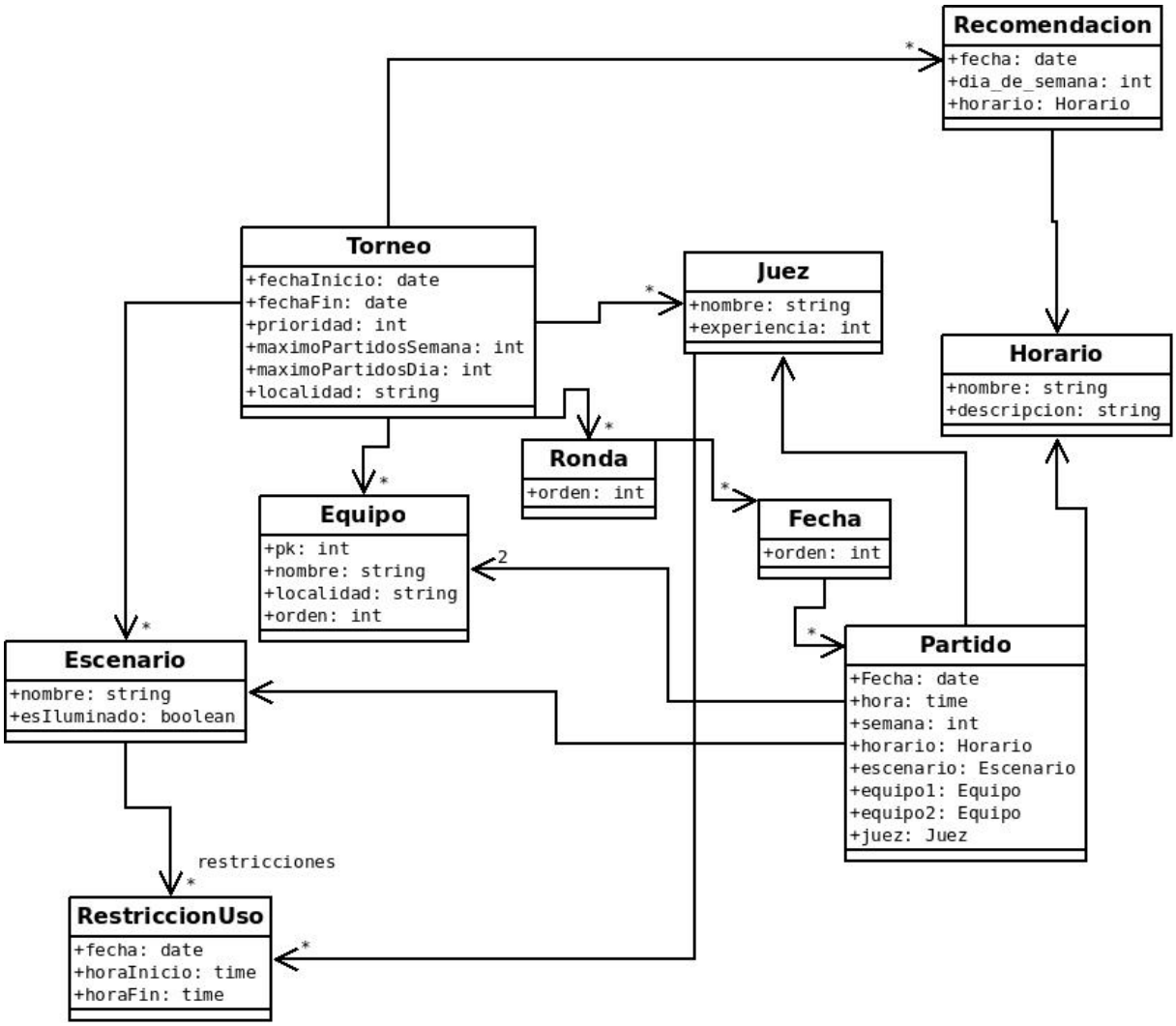

Figura 6.1: Relación entre conceptos relacionados con la organización de torneos de fútbol.

convierten en fuente de inspiración para diseñar el sistema desarrollado bajo este proyecto.

Capítulo 7

ARQUITECTURA DEL SISTEMA

Teniendo en cuenta la especificación de requerimientos, véase apéndice A, se propone la arquitectura de la figura 7.1.

El servicio web se expone a través de un Restful API con Django Rest Framework, Python 2.7. La parte de programación por restricciones para resolver el problema de organizar los partidos del torneo, se ofrece a través de una aplicación escrita en C++, Standard 14, la cuál se puede acceder desde la interfaz de línea de comandos (CLI). Para comunicar estas dos partes del sistema, desde el Restful API en python se llama nuestra aplicación en C++ a través del CLI, enviando parámetros de entrada y obteniendo la solución en el standard input/output, formato JSON.

La aplicación en C++ dependerá de la librería ThorsSerializer para realizar la lectura y escritura de datos en formato JSON, y de las librerías de GECODE para solucionar el modelo de restricciones.

En las siguientes secciones se dará una breve descripción de las tecnologías seleccionadas.

7.1. GECODE

En el grupo de investigación AVISPA, Universidad del Valle, quienes han servido de apoyo para el desarrollo de este proyecto, se ha preferido trabajar tradicionalmente con Mozart OZ para afrontar problemas de satisfacción de restricciones. En este sentido Mozart OZ sería la alternativa de preferencia para agilizar el desarrollo de este proyecto.

Sin embargo Mozart/Oz presenta la desventaja de que en la actual versión no tiene soporte para realizar programación por restricciones [19]. Este componente de programación por restricciones será adicionado a Mozart en el futuro, usando GECODE como apoyo, según el sitio oficial [19].

Obligados de este modo a usar la versión antigua de Mozart, otras desventajas son:

- No disponibilidad de librerías para trabajar con formatos de texto modernos, como JSON.
- No disponibilidad de soporte para trabajar con arquitecturas de computador con multiprocesadores.

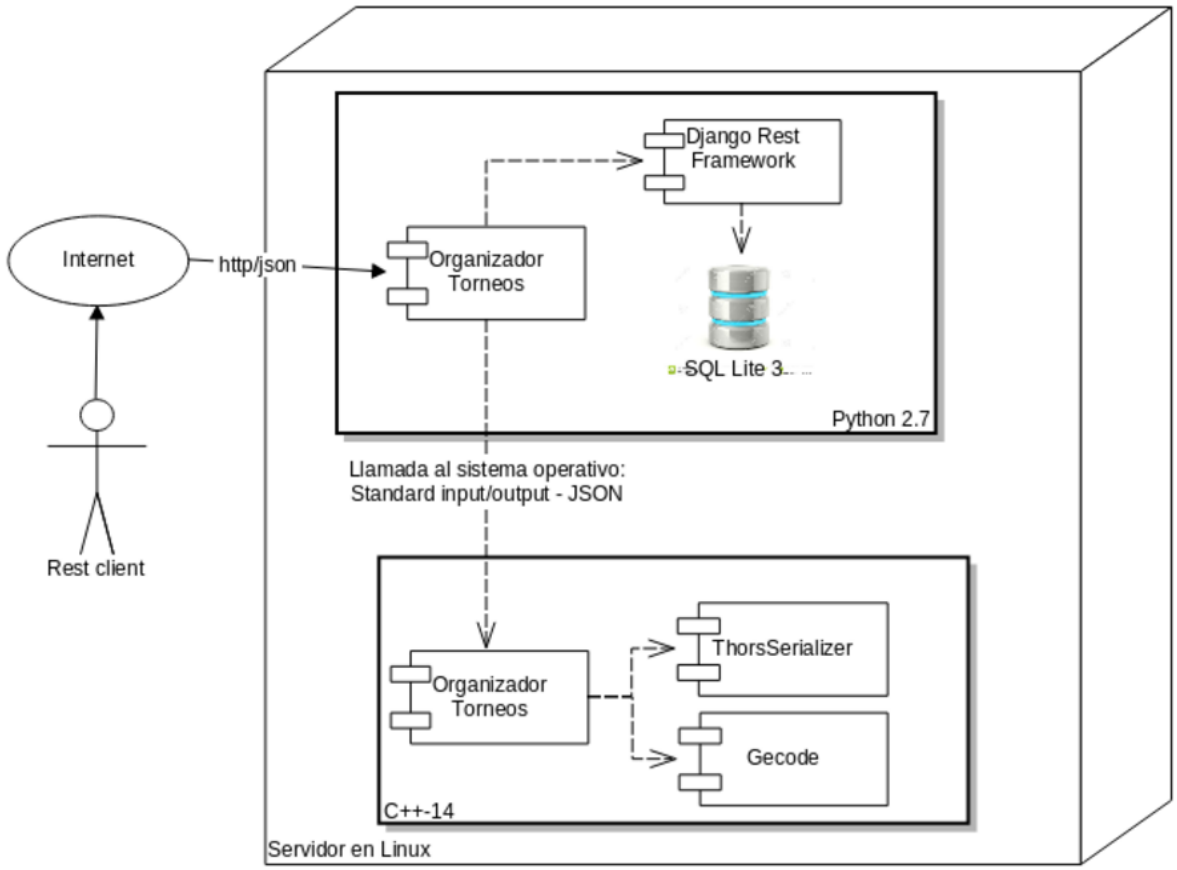

Figura 7.1: Arquitectura del sistema COPISTA.

Dada la desventajas de trabajar con Mozart/Oz y la emergente comunidad alrededor de GECODE, se adoptó este último en el proyecto COPISTA como generador de soluciones. De esta manera se pretende ganar un mejor rendimiento de la aplicación[1] y una mayor facilidad en la implementación y mantenimiento del servicio web.

Otra ventaja de trabajar con GECODE es que supone el desafío de aprender una nueva herramienta, generando así nuevo conocimiento que se puede incorporar en AVISPA.

Un tutorial completo de GECODE se encuentra en [13], en el cual se introduce el uso de GECODE en C++, el modo de instalación y algunos casos de uso como SEND+MORE=MONEY y el problema MAGIC SEQUENCE.

7.2. Django REST Framework

Django REST es el framework que se está usando actualmente en el proyecto SIND, para dar soporte a servicios web de una forma rápida y ligera. Con el fin de mantener la armonía de librerías utilizadas en el proyecto general SIND, se toma la decisión de usarlo en el proyecto, desarrollando entonces nuestro RESTful API en Python.

El modo de instalación y un tutorial de como se usar este framework se puede encontrar en [20].

[1] En la documentación de GECODE se habla de las capacidades del uso de los diferentes procesadores de los computadores modernos para la realización de computación paralela

7.3. Comunicación entre GECODE y Django

Teniendo resuelto cómo se va a implementar la interfaz del servicio web y cómo se va a resolver el problema de satisfacción de restricciones, queda pendiente el cómo resolver la comunicación entre GECODE y Django.

De acuerdo al sitio web de GECODE, actualmente existe una interfaz de gecode para Python [21]. Sin embargo revisando el sitio del proyecto [22] se detectó que: 1) este proyecto aún se encuentra en fase de desarrollo y 2) no se han resuelto bugs del proyecto desde el 2012.

Debido a esto, se implementó la integración entre Python y GECODE utilizando los llamados a comandos del sistema operativo desde Python. De este modo se llama el programa que ejecuta el generador de soluciones de GECODE y se consume los datos que este genere desde un script desarrollado en Python.

Promoviendo la armonía con las tecnologías usadas en el proyecto, se usa JSON como formato de entrada y salida.

Un problema adicional fue el cómo implementar el paso de parámetros y la lectura de la solución. Para esto se tuvieron en cuenta las siguientes alternativas:

1. Uso de archivos de texto. Desde python escribir un archivo de texto con los datos de entrada en formato JSON. Después llamar la ejecución de la aplicación escrita en C++, la cual deberá leer el archivo de texto creado anteriormente, procesar estos datos de entrada y finalmente escribir un escribir un archivo de salida en formato JSON. Finalmente desde python se deberá leer este archivo de salida para poder presentar los resultados. Se requiere un parser de JSON para C++.

2. Uso del Restful Api. Desde python llamar la ejecución del programa escrito en C++, la cual deberá consultar la Restful Api que se está ejecutando en python para traer los datos de entrada, procesar los datos y finalmente escribir la solución en la Restful Api, con lo cual estaría inmediatamente la solución disponible en python para ser presentada al usuario. Para esta solución se requiere el uso de un rest client en c++ que sea capaz de comunicarse con la interfaz expuesta por Django rest framework.

3. Standard Input/Output. Desde python llamar la ejecución del programa escrito en C++ y luego escribir en el Standard Input los datos de entrada en formato JSON. El programa en C++ procesa los datos y escribe la solución en el Standard Output, formato JSON. Los datos de la solución quedan inmediatamente disponibles por la aplicación en python para ser presentados al usuario. Se requiere un parser de JSON para C++.

Para la solución que propone el uso del Restful Api, se encontró la librería restful mapper de C++, la cual es sumamente ligera y simple al usar. A pesar de que esta solución parece una solución limpia, a nivel de arquitectura genera una dependencia desde la aplicación en C++ hacia la aplicación en Django, lo cual hace que nuestra aplicación en C++ no pueda funcionar standalone. Esto va en contra de las buenas practicas de diseño de servicios web [23], en las cuales se promueve que las dependencias entre las diferentes capas del sistema vayan siempre de arriba a abajo, pensando en que las capas que están más arriba son las capas más cercanas al usuario y las capas más bajas son las que proveen servicios como acceso a base de datos o en nuestro caso, una capa (o componente) que ofrece el servicio de generar soluciones a partir de un modelo de programación por restricciones.

Las soluciones que proponen el uso de “Archivos de texto” o el “Standard Input/Output” permiten que la aplicación en C++ sea independiente y ambas soluciones podrían funcionar con un parser sencillo de JSON. Sin embargo se descarta la solución con “Archivos de texto” debido a que 1) representa menor rendimiento al tener que escribir datos en el disco duro, y 2) se debe ejecutar un proceso de limpieza continua porque estos datos deben ser borrados del disco para no tener duplicada una información que cuya persistencia también será administrada por Django.

En conclusión, el uso del “Standard Input/Output” es una solución que se adapta mejor a nuestro proyecto, permitiendo mantener una comunicación directa entre Python y C++, sin crear dependencias desde el componente en C++ hacia el RESTful API.

7.4. ThorsSerializer

ThorsSerializer [24] es una librería sencilla para serializar y des-serializar datos en formato JSON. La curva de aprendizaje e implementación de está librería es muy corta, lo cual permitió agilizar el desarrollo de este proyecto.

7.5. SQLite 3

Dado que la configuración de la base de datos esta fuera del alcance de este proyecto y que SQLite 3, una base de datos relacional minimalista [25], viene configurada por defecto con la instalación de Django Rest Framework, lista para su uso, se deja trabajando el servicio web con los valores por defecto de configuración de base de datos.

Capítulo 8

DESCRIPCIÓN DEL SERVICIO REST

8.1. Diseño de Clases Software

El diagrama de clases del diseño, figura 8.1, pretende mostrar las decisiones de diseño y relaciones entre clases software que se proponen para dar solución a los requerimientos funcionales, descritos en el apéndice A. Es claro que muchas de estas clases software fueron inspiradas en las relaciones conceptuales

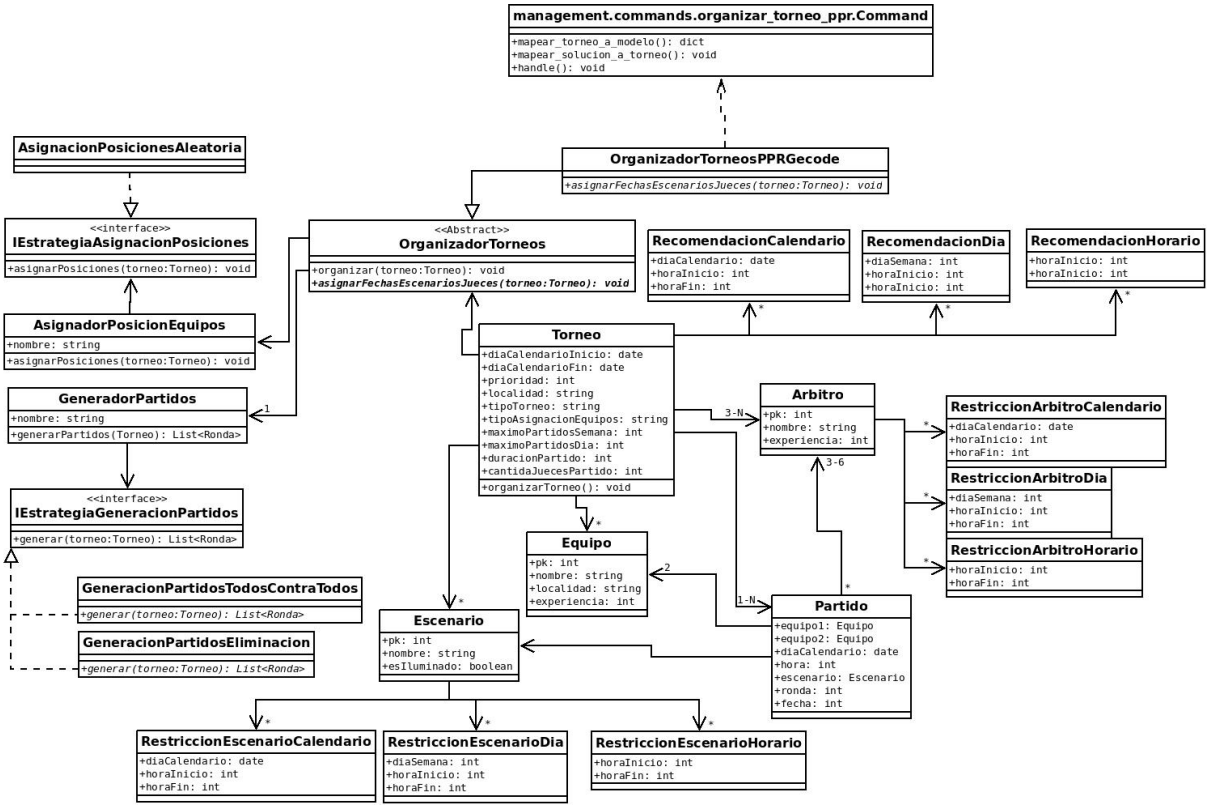

Figura 8.1: Diagrama de clases del Servicio REST.

descritas en el cápitulo 6, logrando de este modo disminuir el salto en representación entre el mundo real y el diseño de nuestra solución software.

Centrando nuestra atención en las relaciones que no aparecen en el diagrama del capítulo 6, podemos observar que un Torneo es responsable de organizarse así mismo mediante un OrganizadorTorneos que implementa el patrón Plantilla[1]. Esta clase abstracta es responsable de asignar u ordenar la posición de los equipos en el torneo y generar los partidos que se deberían ejecutar en el torneo, indicando en que ronda y fecha se juega cada partido. La tarea de asignar el día calendario, hora, jueces y escenario de cada partido debe ser implementada en una clase concreta que extienda del OrganizadorTorneos.

El AsignadorPosicionesEquipos es quien se encarga de ordenar los equipos en el torneo, es decir, indica quién es el equipo 1, 2, 3, etc. Esta clase usa la interfaz IEstrategiaAsignaciónPosiciones (patrón polimorfismo[2]), la cual permitirá implementar a futuro diferentes formas de asignar estas posiciones. Por lo pronto se define la implementación de la clase AsignacionPosicionesAleatoria, la cual ordena los equipos de forma aleatoria. Dada la definición de un torneo, se sabe cuál es la estrategia de organización de equipos mediante el atributo tipoAsignacionTorneo (de tipo string), cuyo único posible valor en esta versión del sistema es “aleatorio”.

Bajo la misma estrategia de diseño de presenta el GeneradorPartidos que usa la interfaz IEstrategiaGeneraciónPartidos. Las clases concretas que definen una estrategia generación son GeneraciónPartidosTodosContraTodos y GeneraciónPartidosEliminación, definiendo así una estrategia para torneos todos contra todos y torneos por eliminación. El sistema sabe qué estrategia de generación de partidos se usa para un torneo en especifico, mediante del atributo tipoTorneo (string), cuyos únicos posibles valores serán “todos contra todos” y “eliminacion”.

La clase concreta OrganizadorTorneosPPRGecode, que extiende el OrganizadorTorneos, adiciona la característica de asignación de días calendario, hora, jueces y escenarios. Esta tarea la lleva a cabo por medio de un llamado asincrónico a un comando personalizado de Django. Mencionado comando es el encargado de mapear los objetos del torneo en un formato reconocible por el generador de soluciones escrito en C++, y luego hace un llamado a este mismo por medio del sistema operativo, enviando la información mapeada por el Standad Input. El comando personalizado de Django queda a la espera hasta que el generador de soluciones de C++ entregue una solución en Standard Output. Una vez leída esta información, el mismo comando se encarga de actualizar los datos de los partidos de acuerdo a la información recibida.

8.2. Operaciones Soportadas por el Servicio

De acuerdo con las clases software de la sección anterior y los requerimientos del apéndice A, la lista a continuación enumera las operaciones y rutas de acceso a dichas operaciones, que soporta el servicio REST.

1. Definición básica de un torneo.

```
url: /api/v1/torneo/
  - método POST: crea un nuevo torneo.
```

[1] Template Method [26]

[2] Polymorphism [27], capítulo 17, sección 11.

```
  - método GET: retorna la lista de torneos registrados.
url: /api/v1/torneo/<id>/
  - método PATCH: modifica el torneo identificado con <id>.
  - método GET: retorna los datos del torneo <id>.
  - método DELETE: elimina el torneo <id>.
formato de los datos:
{
 'dia-calendario-inicio': <date>,
 'dia-calendario-fin': <date>,
 'prioridad': <int>,
 'localidad': <string>,
 'tipo-torneo': ['todos-contra-todos','eliminacion'],
 'tipo-asignacion-equipos': ['aleatoria'],
 'maximo-partidos-semana': <int>,
 'maximo-partidos-dia': <int>,
 'duaracion-partido': <int>,
 'cantidad-jueces-partido': <int>
}
```

2. Definición de las recomendaciones de día calendario para un torneo dado.

```
Url:/api/v1/torneo/<t_id>/recomendacion-dia-calendario/
  - método POST: crea una nueva recomendación para el torneo <t_id>.
  - método GET: retorna la lista de recomendaciones del torneo <t_id>.
Url:/api/v1/torneo/<t_id>/recomendacion-dia-calendario/<id>/
  - método PATCH: modifica la recomendación <id> del torneo <t_id>.
  - método GET: retorna la recomendación <id> del torneo <t_id>.
  - método DELETE: elimina la recomendación <id> del torneo <t_id>.
formato de los datos:
{
 'dia-calendario':<date>,
 'hora-inicial':<int>,
 'hora-final':<int>
}
```

3. Definición de las recomendaciones de día de la semana para un torneo dado.

```
Url:/api/v1/torneo/<t_id>/recomendacion-dia-semana/
  - método POST: crea una nueva recomendación para el torneo <t_id>.
  - método GET: retorna la lista de recomendaciones del torneo <t_id>.
Url:/api/v1/torneo/<t_id>/recomendacion-dia-semana/<id>/
  - método PATCH: modifica la recomendación <id> del torneo <t_id>.
  - método GET: retorna la recomendación <id> del torneo <t_id>.
  - método DELETE: elimina la recomendación <id> del torneo <t_id>.
```

```
formato de los datos:
{
 'dia-semana':<int>,
 'hora-inicial':<int>,
 'hora-final':<int>
}
```

4. Definición de las recomendaciones de horario para un torneo dado.

```
Url:/api/v1/torneo/<t_id>/recomendacion-horario/
  - método POST: crea una nueva recomendación para el torneo <t_id>.
  - método GET: retorna la lista de recomendaciones del torneo <t_id>.
Url:/api/v1/torneo/<t_id>/recomendacion-horario/<id>/
  - método PATCH: modifica la recomendación <id> del torneo <t_id>.
  - método GET: retorna la recomendación <id> del torneo <t_id>.
  - método DELETE: elimina la recomendación <id> del torneo <t_id>.
formato de los datos:
{
 'hora-inicial':<int>,
 'hora-final':<int>
}
```

5. Definición de los árbitros para un torneo dado.

```
Url:/api/v1/torneo/<t_id>/arbitro/
  - método POST: crea un nuevo arbitro para el torneo <t_id>.
  - método GET: retorna la lista de árbitros del torneo <t_id>.
Url:/api/v1/torneo/<t_id>/arbitro/<id>/
  - método PATCH: modifica el árbitro <id> del torneo <t_id>.
  - método GET: retorna el árbitro <id> del torneo <t_id>.
  - método DELETE: elimina el árbitro <id> del torneo <t_id>.
formato de los datos:
{
 'id':<int>,
 'nombre':<string>,
 'experiencia':<int>
}
```

6. Definición de las restricciones de día calendario para un árbitro de un torneo dado.

```
Url:/api/v1/torneo/<t_id>/arbitro/<aid>/restriccion-dia-calendario/
  - método POST: crea una nueva restricción para el árbitro <aid>.
  - método GET: retorna la lista de restricciones del árbitro <aid>.
Url:/api/v1/torneo/<t_id>/arbitro/<aid>/restriccion-dia-calendario/<id>/
```

```
  - método PATCH: modifica la restricción <id>.
  - método GET: retorna la restricción <id>.
  - método DELETE: elimina la restricción <id>.
formato de los datos:
{
 'dia-calendario':<date>,
 'hora-inicial':<int>,
 'hora-final':<int>
}
```

7. Definición de las restricciones de día de la semana para un árbitro de un torneo dado.

```
Url:/api/v1/torneo/<t_id>/arbitro/<aid>/restriccion-dia-semana/
  - método POST: crea una nueva restricción para el árbitro <aid>.
  - método GET: retorna la lista de restricciones del árbitro <aid>.
Url:/api/v1/torneo/<t_id>/arbitro/<aid>/restriccion-dia-semana/<id>/
  - método PATCH: modifica la restricción <id>.
  - método GET: retorna la restricción <id>.
  - método DELETE: elimina la restricción <id>.
formato de los datos:
{
 'dia-semana':<date>,
 'hora-inicial':<int>,
 'hora-final':<int>
}
```

8. Definición de las restricciones de horario para un árbitro de un torneo dado.

```
Url:/api/v1/torneo/<t_id>/arbitro/<aid>/restriccion-horario/
  - método POST: crea una nueva restricción para el árbitro <aid>.
  - método GET: retorna la lista de restricciones del árbitro <aid>.
Url:/api/v1/torneo/<t_id>/arbitro/<aid>/restriccion-horario/<id>/
  - método PATCH: modifica la restricción <id>.
  - método GET: retorna la restricción <id>.
  - método DELETE: elimina la restricción <id>.
formato de los datos:
{
 'hora-inicial':<int>,
 'hora-final':<int>
}
```

9. Definición de los escenarios para un torneo dado.

```
Url:/api/v1/torneo/<t_id>/escenario/
```

```
  - método POST: crea un nuevo escenario para el torneo <t_id>.
  - método GET: retorna la lista de escenarios del torneo <t_id>.
Url:/api/v1/torneo/<t_id>/escenario/<id>/
  - método PATCH: modifica el escenario <id> del torneo <t_id>.
  - método GET: retorna el escenario <id> del torneo <t_id>.
  - método DELETE: elimina el escenario <id> del torneo <t_id>.
formato de los datos:
{
 'id':<int>,
 'nombre':<string>,
 'es-iluminado':<boolean>
}
```

10. Definición de las restricciones de día calendario para un escenario de un torneo dado.

```
Url:/api/v1/torneo/<t_id>/escenario/<eid>/restriccion-dia-calendario/
  - método POST: crea una nueva restricción para el escenario <eid>.
  - método GET: retorna la lista de restricciones del escenario <eid>.
Url:/api/v1/torneo/<t_id>/escenario/<eid>/restriccion-dia-calendario/<id>/
  - método PATCH: modifica la restricción <id>.
  - método GET: retorna la restricción <id>.
  - método DELETE: elimina la restricción <id>.
formato de los datos:
{
 'dia-calendario':<date>,
 'hora-inicial':<int>,
 'hora-final':<int>
}
```

11. Definición de las restricciones de día de la semana para un escenario de un torneo dado.

```
Url:/api/v1/torneo/<t_id>/escenario/<eid>/restriccion-dia-semana/
  - método POST: crea una nueva restricción para el escenario <eid>.
  - método GET: retorna la lista de restricciones del escenario <eid>.
Url:/api/v1/torneo/<t_id>/escenario/<eid>/restriccion-dia-semana/<id>/
  - método PATCH: modifica la restricción <id>.
  - método GET: retorna la restricción <id>.
  - método DELETE: elimina la restricción <id>.
formato de los datos:
{
 'dia-semana':<date>,
 'hora-inicial':<int>,
 'hora-final':<int>
}
```

12. Definición de las restricciones de horario para un escenario de un torneo dado.

```
Url:/api/v1/torneo/<t_id>/escenario/<eid>/restriccion-horario/
  - método POST: crea una nueva restricción para el escenario <eid>.
  - método GET: retorna la lista de restricciones del escenario <eid>.
Url:/api/v1/torneo/<t_id>/escenario/<eid>/restriccion-horario/<id>/
  - método PATCH: modifica la restricción <id>.
  - método GET: retorna la restricción <id>.
  - método DELETE: elimina la restricción <id>.
formato de los datos:
{
 'hora-inicial':<int>,
 'hora-final':<int>
}
```

13. Definición de los equipos para un torneo dado.

```
Url:/api/v1/torneo/<t_id>/equipo/
  - método POST: crea un nuevo equipo para el torneo <t_id>.
  - método GET: retorna la lista de equipos del torneo <t_id>.
Url:/api/v1/torneo/<t_id>/equipo/<id>/
  - método PATCH: modifica el equipo <id> del torneo <t_id>.
  - método GET: retorna el equipo <id> del torneo <t_id>.
  - método DELETE: elimina el equipo <id> del torneo <t_id>.
formato de los datos:
{
 'id':<int>,
 'nombre':<string>,
 'localidad':<string>,
 'experiencia':<int>
}
```

14. Definición de los partidos para un torneo dado[3].

```
Url:/api/v1/torneo/<t_id>/partido/
  - método POST: crea un nuevo partido para el torneo <t_id>.
  - método GET: retorna la lista de partidos del torneo <t_id>.
Url:/api/v1/torneo/<t_id>/partido/<id>/
  - método PATCH: modifica el partido <id> del torneo <t_id>.
  - método GET: retorna el partido <id> del torneo <t_id>.
  - método DELETE: elimina el partido <id> del torneo <t_id>.
formato de los datos:
```

[3] Aunque el sistema puede generar los partidos automáticamente, se deja la posibilidad de adicionar nuevos partidos al torneo a través de la interfaz del REST

```
{
 'equipo-1':<equipo>,
 'equipo-2':<equipo>,
 'dia-calendario':<date>,
 'hora':<int>,
 'escenario':<escenario>,
 'arbitros':<list>(<arbitro>),
 'ronda':<int>,
 'fecha':<int>
}
```

15. Solicitar la generación y organización de los partidos de un torneo.

```
Url:/api/v1/torneo/<id>/solucion/
  - método GET: genera la solicitud para el torneo <id>.
formato de los datos retornados:
{
'response': ['ok', 'fail']
}
```

8.3. Implementación

Como bien se explicó en el capítulo 7, sobre las decisiones tomadas a nivel de arquitectura, el servicio REST se implementó con python y DJango REST Framework. La siguiente es la estructura de archivos en la que se organizó la aplicación Django:

```
- copista.com
    - copista_rest
        - settings.py
        - url.py
    - modelo
        - management
            - commands
                - organizador_torneo_ppr.py
        - admin.py
        - models.py
        - serializers.py
        - views.py
    - management.py
```

En el directorio raíz, copista.com, se encuentra el archivo management.py, con el cual se puede iniciar la aplicación Django y crear la base de datos, entre otras utilidades.

En el directorio copista_rest se encuentra settings.py, donde se encuentra la configuración global del servicio. El archivo url.py define las rutas y operaciones soportadas por el servicio REST, tal como se describió en la sección 8.2.

En el directorio modelo se define el núcleo del sistema desarrollado bajo este proyecto. En el archivo models.py se implementan las clases software mencionadas en la sección 8.1. En el archivo serializers.py se definen los atributos que se van a serializar, a formato JSON, de las clases software implementadas en models.py. El archivo admin.py registra las clases que se quieren tener disponibles en la interfaz web Django de administración de objetos persistentes del sistema. El archivo views.py implementa el cómo van a ser procesadas las peticiones a las operaciones y rutas definidas en el archivo url.py.

En el directorio management.commands se encuentra el archivo organizador_torneo_ppr.py, el cual implementa la clase Command encargada de hacer el llamado a la aplicación que genera las soluciones, como se explicó en la sección 8.1.

Capítulo 9

DESCRIPCIÓN DEL GENERADOR DE SOLUCIONES

9.1. Introducción

De acuerdo a lo planteado en la definición del proyecto, el generador de soluciones implementado se basa en el paradigma de la programación por restricciones, por lo cual se requiere definir el problema en términos de un problema de satisfacción de restricciones (Modelo CSP[1]) e indicar la estrategia de búsqueda de soluciones. En los capítulos siguientes se podrá encontrar ambas definiciones y además una descripción de cómo se implementó la solución en C++ usando GECODE como librería para programación por restricciones.

9.2. Modelo CSP

En esta sección se describe el modelo CSP, diseñado para el problema de organización de torneos de fútbol, teniendo en cuenta los conceptos descritos en el capítulo 6. En las siguientes secciones de describirán los parámetros del modelo, es decir, variables cuyos valores se conocen a priori y que se utilizan como punto de partida para solucionar el problema. Más adelante se describirán las variables de decisión, las cuales en últimas son las que definen cuál fue la solución encontrada. Finalmente se describen la función objetivo y las restricciones que relacionan parámetros y variables de decisión.

9.2.1. Parámetros

- Parámetros que modelan los días del calendario:
  - n: Número de días calendario disponibles para ejecutar el torneo.
  - ns: Número de semanas disponibles para ejecutar el torneo.

[1] Constraint Satisfaction Problem.

- dS_i: Semana a la que pertenece el día calendario i.
  $0 \leq dS_i \leq ns - 1, \forall i = 0, ..., n - 1.$
- dDS_i: Día de la semana a la que pertenece el día calendario i.
  $0 \leq dDS_i \leq 6, \forall i = 0, ..., n - 1.$

- parámetros que describen los días que son recomendados o sugeridos, es decir, días calendario en las cuales se prefiere que se jueguen los partidos:
  - nrd: Número de recomendaciones de día calendario.
  - rdD_{r_d}: Día calendario de la recomendación r_d.
    $0 \leq rdD_{r_d} \leq n - 1, \forall r_d = 0, ..., nrd - 1.$
  - $rdH1_{r_d}$: Hora inicio de la recomendación r_d.
    $0 \leq rdH1_{r_d} \leq 23, \forall r_d = 0, ..., nrd - 1.$
  - $rdH2_{r_d}$: Hora fin de la recomendación r_d.
    $1 \leq rdH2_{r_d} \leq 24, \forall r_d = 0, ..., nrd - 1.$

- parámetros que describen los días de las semana que son recomendados o sugeridos, es decir, días de la semana en los cuales se prefiere que se jueguen los partidos:
  - $nrds$: Número de recomendaciones de día de la semana.
  - $rdsDS_{rds}$: Día de la semana de la recomendación rds.
    $0 \leq rdsDS_{rds} \leq 6, \forall rds = 0, ..., nrds - 1.$
  - $rdsH1_{rds}$: Hora inicio de la recomendación rds.
    $0 \leq rdsH1_{rds} \leq 23, \forall rds = 0, ..., nrds - 1.$
  - $rdsH2_{rds}$: Hora fin de la recomendación rds.
    $1 \leq rdsH2_{rds} \leq 24, \forall rds = 0, ..., nrds - 1.$

- parámetros que describen los horarios que son recomendados o sugeridos, es decir, horarios en los cuales se prefiere que se jueguen los partidos:
  - nrh: Número de recomendaciones de hora.
  - $rhH1_{rh}$: Hora inicio de la recomendación rh.
    $0 \leq rhH1_{rh} \leq 23, \forall rh = 0, ..., nrh - 1.$
  - $rhH2_{rh}$: Hora fin de la recomendación rh.
    $1 \leq rhH2_{rh} \leq 24, \forall rh = 0, ..., nrh - 1.$

- parámetros que modelan los árbitros disponibles y sus atributos:
  - na: Número de árbitros disponibles.
  - aPK_a: Identificador del árbitro a.
    $aPK_a \in \mathbb{N}; \forall a = 0, ..., na - 1.$

- aN_a: Cadena de texto que representa el nombre del árbitro a.
  $\forall a = 0, ..., na - 1$.
- aE_a: Experiencia del árbitro a.
  $aE_a \in \{1, ..., 10\}; \forall a = 0, ..., na - 1$.
- Restricciones de día calendario para los árbitros:
  - $anrd_a$: Número de restricciones de día calendario del árbitro a.
    $\forall a = 0, ..., na - 1$.
  - $ardD_{a,ar_d}$: Día calendario de la restricción ar_d del árbitro a.
    $0 \leq ardD_{a,ar_d} \leq n - 1, \forall a = 0, ..., na - 1, \forall ar_d = 0, ..., anrd_a - 1$.
  - $ardH1_{a,ar_d}$: Hora inicio de la restricción ar_d del árbitro a.
    $0 \leq ardH1_{a,ar_d} \leq 23, \forall a = 0, ..., na - 1, \forall ar_d = 0, ..., anrd_a - 1$.
  - $ardH2_{a,ar_d}$: Hora fin de la restricción ar_d del árbitro a.
    $1 \leq ardH2_{a,ar_d} \leq 24, \forall a = 0, ..., na - 1, \forall ar_d = 0, ..., anrd_a - 1$.
- Restricciones en días de la semana de los árbitros:
  - $anrds_a$: Número de restricciones de día de la semana del árbitro a.
  - $ardsDS_{a,ards}$: Día de la semana de la restricción $ards$ del árbitro a.
    $0 \leq ardsDS_{a,ards} \leq 6, \forall a = 0, ..., na - 1, \forall ards = 0, ..., anrds_a - 1$.
  - $ardsH1_{a,ards}$: Hora inicio de la restricción $ards$ del árbitro a.
    $0 \leq ardsH1_{a,ards} \leq 23, \forall a = 0, ..., na - 1,$
    $\forall ards = 0, ..., anrds_a - 1$.
  - $ardsH2_{a,ards}$: Hora fin de la restricción $ards$ del árbitro a.
    $1 \leq ardsH2_{a,ards} \leq 24, \forall a = 0, ..., na - 1,$
    $\forall ards = 0, ..., anrds_a - 1$.
- Restricciones en horarios de los árbitros:
  - $anrh_a$: Número de restricciones de hora del árbitro a.
  - $arhH1_{a,arh}$: Hora inicio de la restricción arh del árbitro a.
    $0 \leq arhH1_{a,arh} \leq 23, \forall a = 0, ..., na - 1, \forall arh = 0, ..., anrh_a - 1$.
  - $arhH2_{a,arh}$: Hora fin de la restricción arh del árbitro a.
    $1 \leq arhH2_{a,arh} \leq 24, \forall a = 0, ..., na - 1, \forall arh = 0, ..., anrh_a - 1$.

- parámetros que modelan los escenarios disponibles y sus atributos:
  - $nesc$: Número de escenarios disponibles.
  - $escPK_{esc}$: Identificador del escenario esc.
    $\forall esc = 0, ..., nesc - 1$.
  - $escN_{esc}$: Nombre del escenario esc.
    $\forall esc = 0, ..., nesc - 1$.
  - $escI_{esc}$: Indica con 0, que el escenario esc no está iluminado.
    $escI_{esc} \in \{0, 1\}, \forall esc = 0, ..., nesc - 1$.

- Restricciones de día calendario para los escenarios:
  - $escnrd_{esc}$: Número de restricciones de día calendario del escenario esc.
    $\forall esc = 0, ..., nesc - 1$.
  - $escrdD_{esc,escr_d}$: Día calendario de la restricción $escr_d$ del escenario esc.
    $0 \leq escrdD_{esc,escr_d} \leq n - 1, \forall esc = 0, ..., nesc - 1,$
    $\forall escr_d = 0, ..., escnrd_{esc} - 1$.
  - $escrdH1_{esc,escr_d}$: Hora inicio de la restricción $escr_d$ del escenario esc.
    $0 \leq escrdH1_{esc,escr_d} \leq 23, \forall esc = 0, ..., nesc - 1,$
    $\forall escr_d = 0, ..., escnrd_{esc} - 1$.
  - $escrdH2_{esc,escr_d}$: Hora fin de la restricción $escr_d$ del escenario esc.
    $1 \leq escrdH2_{esc,escr_d} \leq 24, \forall esc = 0, ..., nesc - 1,$
    $\forall escr_d = 0, ..., escnrd_{esc} - 1$.
- Restricciones en días de la semana de los escenarios:
  - $escnrds_{esc}$: Número de restricciones de día de la semana del escenario esc.
  - $escrdsDS_{esc,escrds}$: Día de la semana de la restricción $escrds$ del escenario esc.
    $0 \leq escrdsDS_{esc,escrds} \leq 6, \forall esc = 0, ..., nesc - 1,$
    $\forall escrds = 0, ..., escnrds_{esc} - 1$.
  - $escrdsH1_{esc,escrds}$: Hora inicio de la restricción $escrds$ del escenario esc.
    $0 \leq escrdsH1_{esc,escrds} \leq 23, \forall esc = 0, ..., nesc - 1,$
    $\forall escrds = 0, ..., escnrds_{esc} - 1$.
  - $escrdsH2_{esc,escrds}$: Hora fin de la restricción $escrds$ del escenario esc.
    $1 \leq escrdsH2_{esc,escrds} \leq 24, \forall esc = 0, ..., nesc - 1,$
    $\forall escrds = 0, ..., escnrds_{esc} - 1$.
- Restricciones en horarios de los escenarios:
  - $escnrh_{esc}$: Número de restricciones de hora del escenario esc.
  - $escrhH1_{esc,escrh}$: Hora inicio de la restricción $escrh$ del escenario esc.
    $0 \leq escrhH1_{ij} \leq 23, \forall esc = 0, ..., nesc - 1,$
    $\forall escrh = 0, ..., escnrh_{esc} - 1$.
  - $escrhH2_{ij}$: Hora fin de la restricción $escrh$ del escenario esc.
    $1 \leq escrhH2_{ij} \leq 24, \forall esc = 0, ..., nesc - 1,$
    $\forall escrh = 0, ..., escnrh_{esc} - 1$

- parámetros que modelan los equipos disponibles y sus atributos:
  - ne: Número de equipos.
  - ePK_e: Identificador del equipo e.
    $\forall e = 0, ..., ne - 1$.
  - eN_e: Nombre del equipo e.
    $\forall e = 0, ..., ne - 1$.
- Parámetros que definen las rondas del torneo:

- nr: Número de rondas del torneo.
- rnf_r: Número de fechas en la ronda r.
  $\forall r = 0, ..., nr - 1$.

- Parámetros que definen las fechas del torneo:
  - nf: Número total de fechas en el torneo.
  - fnp_f: Número de partidos de la fecha f.
    $\forall f = 0, ..., nf - 1$.
  - fR_f: Ronda a la que pertenece la fecha f.
    $0 \leq fR_f \leq nr - 1, \forall f = 0, ..., nf - 1$.

- Parámetros que definen los partidos del torneo:
  - np: Número de partidos del torneo.
  - $pEQ1_p$: Equipo 1 del partido p.
    $0 \leq pEQ1_p \leq ne - 1, \forall p = 0, ..., np - 1$.
  - $pEQ2_p$: Equipo 2 del partido p.
    $0 \leq pEQ2_p \leq ne - 1, \forall p = 0, ..., np - 1$.
  - pF_p: Fecha a la que pertenece el partido p.
    $0 \leq pF_p \leq nf - 1, \forall p = 0, ..., np - 1$.
  - pR_p: Ronda a la que pertenece el partido p.
    $pR_p = fR_{pF_p}, \forall p = 0, ..., np - 1$.

- Otros parámetros que describen atributos del torneo:
  - mps: Máxima cantidad de partidos permitidas para jugar en una semana.
  - mpd: Máxima cantidad de partidos permitidas para jugar en un día.
  - dp: Duración de cada partido.
  - cap: Cantidad de árbitros por partido.

9.2.2. Variables de decisión

- pD_p: Dia calendario del partido p.
  $0 \leq pD_p \leq n - 1, \forall p = 0, ..., np - 1$.
- pH_p: Hora de inicio del partido p.
  $0 \leq pH_p \leq 23, \forall p = 0, ..., np - 1$.
- $pESC_p$: Escenario del partido p.
  $-1 \leq pESC_p \leq nesc - 1, \forall p = 0, ..., np - 1$.
  Tener -1 en $pESC_p$ significa que no fue asignado un escenario para el partido p.

- $pAP_{ap,p}$: Indica quien es el ap-esimo árbitro del partido p.

  $-1 \leq pAP_{ap,p} \leq na - 1, \forall p = 0, ..., np - 1$, and $\forall ap = 0, ..., cap - 1$.

  Tener -1 en $pAP_{ap,p}$ significa que el ap-esimo árbitro del partido p no fue asignado.

9.2.3. Variables auxiliares de decisión

- pS_p: Semana del partido p.

  $0 \leq pS_p \leq ns - 1, \forall p = 0, ..., np - 1$.

- pDS_p: Día de la semana del partido p.

  $0 \leq pDS_p \leq 6, \forall p = 0, ..., np - 1$.

- $pA_{a,p}$: Indica si el árbitro a juzga en el partido p.

  $pA_{a,p} \in \{0, 1\}, \forall a = 0, ..., na - 1, \forall p = 0, ..., np - 1$.

- $cESCnd$: Cantidad de escenarios no definidos de todos los partidos.

  $cESCnd \in \{0, np\}$.

- $cAPnd$: Cantidad de árbitros no definidos de todos los partidos.

  $cAPnd \in \{0, cap \times np\}$.

- cHR: Cantidad de horas de los partidos, jugadas en las horas de recomendaciones de día calendario, día de la semana u hora.

- cAE: Cuantificador de cuán bien se asignaron los árbitros al torneo. La política de asignación utilizada es: en cada partido los árbitros asignados son aquellos con mayor experiencia posible, teniendo en cuenta que a medida que aumentan las rondas en el torneo, los partidos de dichas rondas tienen mayor prioridad para recibir dichos árbitros.

- fo: Función objetivo, la cual se definirá en términos de las variables cHR, cAE, $cESCnd$ y $cAPnd$.

9.2.4. Función objetivo

El generador de soluciones debe elegir la mejor solución entre las soluciones encontradas. La función objetivo es una función que nos ayuda a establecer una medida con la cual definir cuando una solución es mejor que otra.

En nuestro problema particular, se puede decir que una solución A es mejor que otra B, para un torneo dado, cuando la solución A:

- tiene mayor cantidad de horas en que los partidos se están jugando dentro de un rango de horas recomendadas, lo cual se define en la variable cHR.

- tiene asignado los árbitros a los partidos de tal manera que los árbitros con mayor experiencia se asignan preferiblemente a las rondas finales. Esto se representa mediante la variable cAE.

- tiene menos partidos a los que no se las haya podido asignar escenario. La variable $cESCnd$ nos ayuda a contabilizar a cuantos partidos no se les ha asignado escenario.
- tiene menor valor en la variable $cAPnd$, es decir tiene menor cantidad de árbitros no definidos en todos los partidos.

Una primera intuición de la función objetivo sería:
$Maximizar fo = cHR + cAE - cESCnd - cAPnd$
Sin embargo, estos cuatro objetivos no son independientes y puede darse casos en los que dar prioridad a un objetivo significa sacrificar otros objetivos. Por lo tanto se hace necesario definir una prioridad a estos objetivos.

Los siguientes enunciados establecen la prioridad entre objetivos:

- programar un partido en los horarios recomendados debe ser más importante que programar los árbitros con mejor experiencia.
- tener programado los árbitros del partido es mucho más importante que jugar los partidos en el horario recomendado.
- asignar un escenario a un partido es tan importante como tener asignado los cap árbitros del mismo.

Partiendo del hecho de que las experiencias de los árbitros están en un rango entre 1 y 10, se podría dar más importancia a jugar los partidos en horarios recomendados, dando un peso al cumplimiento de las recomendaciones con un valor por encima de 10.

Las recomendaciones de horarios tienen tres tipos de recomendaciones: la de horario, la de día de la semana y la de día calendario, donde la recomendación de día calendario es la más importante de las tres y la de horarios es la de menor importancia. Siguiendo lo mencionado en el anterior párrafo, se define que las recomendaciones de horario tiene un peso de 11, las de día de la semana tienen 13 y las de día calendario 15.

Tener asignado escenario y árbitros en los partidos es mucho más importante que todo lo demás, entonces se ajusta la función objetivo fo, para que el disminuir $cESCnd$ y $cAPnd$ sea 100 veces más importante. La siguiente relación algebraica muestra lo anteriormente dicho:
$Maximizar fo = cHR + cAE - 100 \times cESCnd - 100 \times cAPnd$
El último enunciado, acerca de la importancia de tener asignados los cap árbitros del partido respecto a tener asignado el escenario del mismo, transforma la función objetivo de la siguiente manera:
$Maximizar fo = cHR + cAE - 100 \times cap \times cESCnd - 100 \times cAPnd$
A continuación se define formalmente la función objetivo y sus variables relacionadas:

- $Maximizar fo = cHR + cAE - 100 \times cap \times cESCnd - 100 \times cAPnd$

  donde:

  $cESCnd = |\{p : pESC_p = -1, 0 \leq p \leq np - 1\}|$

  $cAPnd = |\{ap, p : pAP_{ap,p} = -1, 0 \leq ap \leq cap - 1, 0 \leq p \leq np - 1\}|$

  $$cAE = \sum_{\substack{0 \leq a \leq na-1 \\ 0 \leq p \leq np-1}} pA_{a,p} \times aE_a \times fR_{pF_p}$$

$$cHR = 15 \sum_{\substack{0 \leq r_d \leq nrd-1 \\ 0 \leq p \leq np-1}} cHRD_{r_d,p} + 13 \sum_{\substack{0 \leq rds \leq nrds-1 \\ 0 \leq p \leq np-1}} cHRDS_{rds,p} + 11 \sum_{\substack{0 \leq rh \leq nrh-1 \\ 0 \leq p \leq np-1}} cHRH_{rh,p}$$

$cHRD_{r_d,p}$: es la cantidad de horas que en el partido p se jugaron en el rango de tiempo de la recomendación de fecha r_d.

$cHRD_{r_d,p} = max(0, mindT2_{r_d,p} - maxdT1_{r_d,p})$

$maxdT1_{r_d,p} = max(rdT1_{r_d}, pdT1_p)$

$mindT2_{r_d,p} = min(rdT2_{r_d}, pdT2_p)$

$rdT1_{r_d} = 24 \times rdD_{r_d} + rdH1_{r_d}$

$rdT2_{r_d} = 24 \times rdD_{r_d} + rdH2_{r_d}$

$pdT1_p = 24 \times pD_p + pH_p$

$pdT2_p = pdT1_p + dp$

$cHRDS_{rds,p}$: es la cantidad de horas que en el partido p se jugaron en el rango de tiempo de la recomendación de día de la semana rds.

$cHRDS_{rds,p} = max(0, mindsT2_{rds,p} - maxdsT1_{rds,p})$

$maxdsT1_{rds,p} = max(rdsT1_{rds}, pdsT1_p)$

$mindsT2_{rds,p} = min(rdsT2_{rds}, pdsT2_p)$

$rdsT1_{rds} = 24 \times rdsF_{rds} + rdsH1_{rds}$

$rdsT2_{rds} = 24 \times rdsF_{rds} + rdsH2_{rds}$

$pdsT1_p = 24 \times pDS_p + pH_p$

$pdsT2_p = pdsT1_p + dp$

$cHRH_{rh,p}$: es la cantidad de horas que en el partido p se jugaron en el rango de tiempo de la recomendación de horario rh.

$cHRH_{rh,p} = max(0, minhT2_{rh,p} - maxhT1_{rh,p})$

$maxhT1_{rh,p} = max(rhH1_{rh}, pH_p)$

$minhT2_{rh,p} = min(rhH2_{rh}, phT1_p + dp)$

9.2.5. Restricciones

- Restricción 1: Relaciona la semana y día de la semana de un partido con el día calendario en que se juega ese mismo partido.

  $pS_p = dS_{pD_p}$

  $pDS_p = dDS_{pD_p}$

  $\forall p = 0, ..., np - 1$

- Restricción 2: Ordena ascendentemente los días de los partidos de una misma fecha (permitiendo que estos se puedan jugar en el misma día calendario) y asegura que los partidos de fechas y rondas diferentes, no se jueguen el mismo día.

$\forall(p1, p2) \in \{0, ..., np - 1\}$

$p1 < p2 \land pF_{p1} = pF_{p2} \implies pD_{p1} \leq pD_{p2}$

$pF_{p1} < pF_{p2} \implies pD_{p1} < pD_{p2}$

$pR_{p1} < pR_{p2} \implies pD_{p1} < pD_{p2}$

- Restricción 3: Impedir que un equipo juegue más de un partido por día.

  $\forall(p1, p2) \in \{0, ..., np - 1\}, \forall a = 0, ..., na - 1$

  $p1 \neq p2 \land \{pEQ1_{p1}, pEQ2_{p1}\} \cap \{pEQ1_{p2}, pEQ2_{p2}\} \neq \emptyset \implies pT2_{p1} \leq pT1_{p2} \lor pT2_{p2} \leq pT1_{p1}$

  Donde:

  $pT1_p = 24 \times pD_p + pH_p$

  $pT2_p = pT1_p + dp$

- Restricción 4: Permitir como máximo mpd partidos en la misma fecha.

  $|\{p : pD_p = i, 0 \leq p \leq np - 1\}| \in \{0, ..., mpd\}$

  $\forall i = 0, ..., n - 1$

- Restricción 5: Permitir como máximo mps partidos con la misma semana.

  $|\{p : pS_p = s, 0 \leq p < np - 1\}| \in \{0, ..., mps\}$

  $\forall s = 0, ..., ns - 1$

- Restricción 6: Impedir que dos partidos se solapen en la misma fecha y hora, sí están asignados al mismo escenario.

  $\forall(p1, p2) \in \{0, ..., np - 1\}$

  $p1 \neq p2 \land pESC_{p1} \neq -1 \land pESC_{p1} = pESC_{p2} \implies$

  $pT2_{p1} \leq pT1_{p2} \lor pT2_{p2} \leq pT1_{p1}$

  Donde:

  $pT1_p = 24 \times pD_p + pH_p$

  $pT2_p = pT1_p + dp$

- Restricción 7: No se puede jugar partidos entre las 00:00 y las 06:00 y entre las 18:00 y 24:00 (noche) en los escenarios donde no hay iluminación.

  $\forall esc = 0, ..., nesc - 1, \forall p = 0, ..., np - 1$

  $escI_{esc} = 0 \land pESC_p = esc \implies 6 \leq pH_p \land pH_p + dp \leq 18$

- Restricción 8: Dado un escenario, no se puede jugar partidos en dicho escenario, en las horas de las restricciones del mismo. Aplica para restricciones de dia calendario, día de la semana u hora.

  $\forall esc = 0, ..., nesc - 1, \forall p = 0, ..., np - 1$

  $pESC_p = esc \implies$

$\forall escr_d = 0, ..., escnrd_{esc} - 1$:

$pdT2_p \leq escrdT1_{esc,escr_d} \vee escrdT2_{esc,escr_d} \leq pdT1_p \wedge$

$\forall escrds = 0, ..., escnrds_{esc} - 1$:

$pdsT2_p \leq escrdsT1_{esc,escrds} \vee escrdsT2_{esc,escrds} \leq pdsT1_p \wedge$

$\forall escrh = 0, ..., escnrh_{esc} - 1$:

$pH_p + dp \leq escrhH1_{esc,escrh} \vee escrhH2_{esc,escrh} \leq pH_p$

Donde:

$pdT1_p = 24 * pD_p + pH_p$

$pfT2_p = pdT1_p + dp$

$escrdT1_{esc,escr_d} = 24 * escrdD_{esc,escr_d} + escrfdH1_{esc,escr_d}$

$escrdT2_{esc,escr_d} = 24 * escrdD_{esc,escr_d} + escrdH2_{esc,escr_d}$

$pdsT1_p = 24 * pS_p + pH_p$

$pdsT2_p = pdsT1_p + dp$

$escrdsT1_{esc,escrds} = 24 * escrdsF_{esc,escrds} + escrdsH1_{esc,escrds}$

$escrdsT2_{esc,escrds} = 24 * escrdsF_{esc,escrds} + escrdsH2_{esc,escrds}$

- Restricción 9: Se deben asignar máximo cap árbitros por cada partido.

  $\forall p = 0, ..., np - 1$

  $|\{a : pA_{a,p} = 1, 0 \leq a \leq na - 1\}| \leq cap$

- Restricción 10: Los partidos a los que se asigne el mismo árbitro no se pueden solapar en fecha y hora.

  $\forall (p1, p2) \in \{0, ..., np - 1\}, \forall a = 0, ..., na - 1$

  $p1 \neq p2 \wedge pA_{a,p1} = 1 \wedge pA_{a,p2} = 1 \implies pT2_{p1} \leq pT1_{p2} \vee pT2_{p2} \leq pT1_{p1}$

  Donde:

  $pT1_p = 24 \times pD_p + pH_p$

  $pT2_p = pT1_p + dp$

- Restricción 11: Los partidos en los que un árbitro juzga, no se pueden jugar en las horas en que dicho árbitro no puede juzgar. Aplica para restricciones de dia calendario, día de la semana u hora del árbitro.

  $\forall a = 0, ..., na - 1, \forall p = 0, ..., np - 1$

  $pA_{a,p} = 1 \implies$

  $\forall ar_d = 0, ..., anrd_a - 1$: $pdT2_p \leq ardT1_{a,ar_d} \vee ardT2_{a,ar_d} \leq pdT1_p \wedge$

  $\forall ards = 0, ..., anrds_a - 1$: $pdsT2_p \leq ardsT1_{a,ards} \vee ardsT2_{a,ards} \leq pdsT1_p \wedge$

  $\forall arh = 0, ..., anrh_a - 1$: $pH_p + dp \leq arhH1_{a,arh} \vee arhH2_{a,arh} \leq pH_p$

  Donde:

$pdT1_p = 24 * pD_p + pH_p$

$pdT2_p = pdT1_p + dp$

$ardT1_{a,ar_d} = 24 * ardD_{a,ar_d} + ardH1_{a,ar_d}$

$ardT2_{a,ar_d} = 24 * ardD_{a,ar_d} + ardH2_{a,ar_d}$

$pdsT1_p = 24 * pS_p + pH_p$

$pdsT2_p = pdsT1_p + dp$

$ardsT1_{a,ards} = 24 * ardsF_{a,ards} + ardsH1_{a,ards}$

$ardsT2_{a,ards} = 24 * ardsF_{a,ards} + ardsH2_{a,ards}$

- Restricción 12: relacionar el ap-esimo arbitro asignado a un partido p, con la variable de seleccion auxiliar que indica si ese arbitro juzga o no un partido.

  $\forall a = 0, ..., na - 1, \forall p = 0, ..., np - 1$:

  $|\{ap : pAP_{ap,p} = a, 0 \leq ap \leq cap - 1\}| \in \{0, 1\}$

  $pA_{a,p} = |\{ap : pAP_{ap,p} = a, 0 \leq ap \leq cap - 1\}|$

9.3. Estrategias de distribución

Se proponen tres estrategias de distribución, las cuales se pueden seleccionar al iniciar el generador de soluciones. A cada estrategia se le asigna un nombre con el cual identificarla: estrategia naive_1, naive_2 y on_problem_1. En las siguientes secciones se puede encontrar una pequeña discusión acerca del funcionamiento de cada una de ellas.

9.3.1. Naive 1

La estrategia naive_1[2] separa las variables en grupos y luego genera un proceso de distribución por cada grupo.

La agrupación de variables se hace por tipo de variable, es decir, el conjunto de variables fecha-hora, un conjunto de variables escenario, y los conjuntos de variables ap-esimo ábitro asignado a cada partido. Entonces cada conjunto se define como sigue:

$T = \{pT1_p : 0 \leq p < np\}$, donde $pT1_p = pD_p + pH_p$

$ESC = \{pESC_p : 0 \leq p < np\}$

$AP_{ap} = \{pAP_{ap,p} : 0 \leq p < np\}, \forall ap = 0, ..., cap - 1$

El primer proceso de distribución se llama para el conjunto de variables ESC, luego para los conjuntos de variables AP_{ap} y finalmente para el conjunto de variables T.

En cada llamado al proceso de distribución:

1. se indica que se seleccione para distribuir primero la variable con menor dominio, es decir:

   $x \coloneqq a_i : |a_i| \leq |a_j|, \forall a_j : 0 \leq j < |A|$

   donde:

[2] En inglés naive significa ingenuo

- x es la variable seleccionada por el distribuidor,
- A es el conjunto de variables a seleccionar,
- a_i y a_j son variables del conjunto A,
- $|a_i|$ es la cantidad de elementos del dominio de la variable a_i, y
- $|A|$ es la cantidad de variables del conjunto A.

2. también se indica que se generen dos ramas de búsqueda, una donde el valor de la variable seleccionada sea el menor valor del dominio de dicha variable y otra rama dónde el valor de dicha variable sea diferente al menor valor de su dominio. Es decir:

   En la rama de búsqueda 1 se adiciona una la restricción:

   $x = min(x)$

   y en la rama de búsqueda 2 se adiciona la restricción:

   $x \neq min(x)$

   donde:

   - x es la variable seleccionada por el distribuidor, y
   - min es una función que retorna el mínimo valor del dominio de la variable x.

9.3.2. Naive 2

En la estrategia naive_2 se hace un único llamado al proceso de distribución con un conjunto global de todas la variables a distribuir.

$G = \{pESC_p, pAP_{ap,p}, pT1_p : 0 \leq p < np \wedge 0 \leq ap < cap\}$

donde:

- G es el conjunto global de todas las variables a ser distribuidas.

Al igual que en la estrategia naive_1, en esta estrategia se indica al proceso de distribución que seleccione la variable con menor cantidad de elementos en el dominio y se crean las dos nuevas ramas de búsqueda basado en si x es o no el mínimo valor del dominio de la variable x.

9.3.3. On problem 1

La estrategia on_problem_1 es una estrategia de distribución basada en ciertas características propias del problema que estamos resolviendo, el proceso de organizar torneos de fútbol. En esta esta estrategia se separan las variables agrupadas por partidos y luego se llama el proceso de distribución para cada conjunto.

Entonces $\forall p : 0 \leq p < np$:

$P_p = \{pESC_p, pAP_{ap,p}, pT1_p : 0 \leq ap < ca\}$

Primero se llama el proceso de distribución para el conjunto P_{np-1}, luego para el conjunto P_{np-2}, luego para el conjunto P_{np-3} y así sucesivamente hasta el conjunto P_0.

En cada llamado al proceso de distribución también se indica:

1. que se seleccione para distribuir la variable que primero fue declarada en el conjunto, es decir:

   $x \coloneqq a_i : i \leq j, \forall j : 0 \leq j < |A|$

   Donde:

   - x es la variable seleccionada por el distribuidor,
   - A es el conjunto de variables a seleccionar,
   - i y j son índices de los elementos de A, y
   - $|A|$ es la cantidad de variables del conjunto A.

2. que se generen las dos nuevas ramas de búsqueda basado en la variable que se seleccionó para ser distribuida. Si la variable seleccionada es para definir el escenario, entonces simplemente se crean dos nuevas ramas de tal forma que la variable seleccionada es o no el valor máximo de su dominio. En caso de que la variable seleccionada es uno de los árbitros del partido, entonces se asigna entre los valores de su dominio, aquel que represente al árbitro con mayor experiencia. Finalmente si la variable a seleccionar es la fecha-hora, se escoge entre los valores de su dominio, aquel que represente una fecha-hora en la que más recomendaciones de día calendario, día de la semana u hora se cumple. Formalmente:

   Sea:

   - a_0 la variable del conjunto A que representa $pESC$, el escenario.
   - a_{cap+1} la variable que representa $pT1$, la fecha hora.
   - $a_1, a_2, ..., a_{cap}$ representan los árbitros del partido, es decir, $pAP_{0,p}, pAP_{1,p}, ..., pAP_{cap-1,p}$.

   Los casos serían:

   - Si $x \coloneqq a_0$, cada posible valor de x representa un escenario y entonces:
     - En la rama de búsqueda 1 se adiciona la restricción:
       $x = max(x)$
     - y en la rama de búsqueda 2 se adiciona la restricción:
       $x \neq max(x)$

     donde: max es una función que retorna el valor máximo entre los elementos del dominio de la variable x.
   - Si $x \coloneqq a_i : i = cap+1$, los valores de x representan una fecha hora $t \coloneqq pT1_p = pD_p + pH_p$ y entonces:
     - En la rama de búsqueda 1 se adiciona la restricción:
       $x = t$
     - y en la rama de búsqueda 2 se adiciona la restricción:
       $x \neq t$

     Donde:
     - $t \coloneqq x_i : tV_{x_i} \geq tV_{x_j} \forall j : 0 \leq j < |x|$
     - x_i y x_j son valores diferentes que pertenecen al dominio de x.

- tV_t se define de la siguiente forma:
  $tV_t = 15 \sum_{0 \leq r_d < nrd} cHRD_{r_d,t} + 13 \sum_{0 \leq rds < |rdsT1|} cHRDS_{rds,t} + 11 \sum_{0 \leq rh < |rhT1|} cHRH_{rh,t}$
- $cHRD_{r_d,t}$ es a cantidad de horas que un partido, que comienza en el tiempo t, comparte con las horas recomendadas de día calendario r_d.
  $cHRD_{r_d,t} = max(0, mindT2_{r_d,t} - maxdT1_{r_d,t})$
  $maxdT1_{r_d,t} = max(rdT1_{r_d}, t)$
  $mindT2_{r_d,t} = min(rdT2_{r_d}, t + dp)$
  $rdT1_{r_d} = 24 \times rdD_{r_d} + rdH1_{r_d}$
  $rdT2_{r_d} = 24 \times rdD_{r_d} + rdH2_{r_d}$
- $cHRDS_{rds,t}$: es la cantidad de horas que un partido que comienza en el tiempo t se juega en el rango de tiempo de la recomendación de día de la semana rds.
  $cHRDS_{rds,t} = max(0, mindsT2_{rds,t} - maxdsT1_{rds,t})$
  $maxdsT1_{rds,t} = max(rdsT1_{rds}, t)$
  $mindsT2_{rds,t} = min(rdsT2_{rds}, t + dp)$
- $rdsT1 = \{24i + rdsH1_{rds} : dDS_i = rdsDS_{rds}, 0 \leq i < n, 0 \leq rds < nrds\}$
- $rdsT2 = \{24i + rdsH2_{rds} : dDS_i = rdsDS_{rds}, 0 \leq i < n, 0 \leq rds < nrds\}$
- $cHRH_{rh,t}$: es la cantidad de horas que un partido que comienza en el tiempo t se juega en el rango de tiempo de la recomendación de horario rh.
  $cHRH_{rh,t} = max(0, minhT2_{rh,t} - maxhT1_{rh,t})$
  $maxhT1_{rh,t} = max(rhT1_{rh}, t)$
  $minhT2_{rh,p} = min(rhT2_{rh}, t + dp)$
- $rhT1 = \{24i + rH1_{rh} : 0 \leq i < n, 0 \leq rds < nrds\}$
- $rhT2 = \{24i + rhH2_{rh} : 0 \leq i < n, 0 \leq rds < nrds\}$
- dp es la duración de los partidos del torneo.
- n es la cantidad de días calendario disponibles para el torneo.
- nrd número de recomendaciones de día calendario.
- $nrds$ cantidad de recomendaciones de día de la semana.
- nrh número de recomendaciones de hora.

- Finalmente, si no cumple con ninguno de los casos anteriores, quiere decir que la variable a distribuir es uno de los árbitros del partido y que los posibles valores del dominio de aquella variable representan un árbitro a. Entonces:

  - En la rama de búsqueda 1 se adiciona la restricción:
    $x = a$
  - y en la rama de búsqueda 2 se adiciona la restricción:
    $x \neq a$

  Donde:

  - $a \coloneqq x_i : aE_{x_i} \geq aE_{x_j} \forall j : 0 \leq j < |x|$.
  - aE_a es la experiencia del árbitro a.

9.4. Implementación

La siguiente estructura de directorios y archivos muestra como está organizado el código fuente del generador de soluciones.

```
- c_solver
    - header_files
        ModeloTorneoFutbol.h
        parametros.h
        solution.h
    - problem_examples
        example_1.json
    - source_files
        ModeloTorneoFutbol.cpp
    Makefile
    principal.cpp
```

El archivo Makefile contiene las instrucciones para construir la aplicación ejecutando el comando $make$. Principal.cpp es el archivo fuente, escrito en C++, que contiene la rutina $main$, la cual orquesta los diferentes componentes de la aplicación para generar las soluciones.

En la carpeta $header\_files$ se encuentran los archivos que declaran las clases que se están usando para generar la solución. $ModeloTorneoFutbol.h$ tiene las declaraciones concernientes a los componentes que interactuann con GECODE para generar las soluciones. El archivo $parametros.h$ define el formato JSON de los parámetros que se van a leer como entrada desde el standard input. Del mismo modo, el archivo $solution.h$ define el formato JSON de salida de las soluciones. Estos dos últimos archivos también declaran las dependencias necesarias de Thorserializer para realizar todo lo relacionado al mapeo de objetos a cadenas de texto o viceversa.

La carpeta $problem\_example$ contiene ejemplos para validar el generador de soluciones con casos conocidos de problemas de organización de torneos.

El directorio $source\_files$ contiene el archivo $ModeloTorneoFutbol.cpp$, el cual define la implementación de las funciones declaradas en $ModeloTorneoFutbol.h$. Entre las principales implementaciones de este archivo se encuentra la implementación de las restricciones, la función objetivo y las estrategías de distribución. En las siguientes secciones se discutirá las decisiones tomadas en estos tres aspectos.

9.4.1. Restricciones

Las restricciones declaradas formalmente en el modelo CSP son una fuente de inspiración para la implementación del generador de soluciones[3]. Sin embargo en algunos casos es posible reducir el problema que se pretendía solucionar con la restricción que se declaró, en términos de propagadores que ya ofrece implementados GECODE. También existe un caso en el cual la restricción declarada no se implementó, debido al grado de dificultad de la implementación, las limitaciones en tiempo del proyecto para investigar y porque resultó no ser una restricción prioritaria, debido a que otras restricciones del CSP ya se encargaban de satisfacer la relación.

[3] Al igual que el diagrama de clases conceptuales inspira el diseño de clases software.

A continuación se describen los casos especiales de implementación de las restricciones declaradas en el CSP:

- El objetivo de la restricción 3 es impedir que un equipo juegue más de un partido por día. La forma en que se declaró la restricción implica la operación entre conjuntos de variables. A pesar de que GECODE ofrece herramientas para crear restricciones sobre conjuntos, esta implementación puede ser difícil de realizar. Entonces se decidío dejar la implementación de esta restricción como trabajo futuro, puesto que la restricción dos[4] satisface los objetivos de la restricción 3, dado que los equipos tienen solo un partido por fecha, tanto en torneos por eliminatoria como en torneos todos contra todos.

- En la restricción 6, acerca de no permitir que dos partidos se asignen al mismo escenario en un mismo día y hora calendario, se encontró que se podía reducir la definición de la restricción a un problema de scheduling[5].

  GECODE ofrece diferentes propagadores para resolver problemas de scheduling. Por ejemplo el propagador $cumulatives$ [28] [29], el cual se encarga de organizar tareas en el tiempo y distribuirlas entre diferentes máquinas que las resuelven. A las tareas se les puede definir cuánto tiempo y cuánto recurso de una máquina requiere para ser terminada. A las máquinas se les puede definir cuánto recurso tienen disponible para solucionar cada tarea.

  Si definimos nuestro modelo en términos del propagador $cumulatives$, cada escenario sería una máquina y los partidos serían las tareas a resolver. Los escenarios pueden atender solo a un partido por vez, es decir, cada máquina puede resolver una tarea por vez. Cada partido ocupa todo el escenario cuando se está jugando, es decir, cada tarea ocupa todo el recurso de una máquina mientras se está ejecutando. De los partidos también podemos decir que cada uno dura dp horas, es decir, cada tarea demora dp horas en ser terminada.

- La restricción 7, acerca de no permitir jugar partidos antes de las 6 de la mañana o después de las 6 de la tarde en los escenarios que no están iluminados, es tratada como una restricción de horario. Por lo tanto se añade a las restricciones de horario de los escenarios no iluminados, no jugar en los mencionados rangos de horario.

- La restricción 8, acerca de los día calendario, días de la semana y horas en que no puede ser usado un escenario, también es un ejemplo de reducción de problemas. En este caso se identificó que las restricciones de uso de escenario pueden ser interpretadas también como tareas a propagar con $cumulatives$, donde este tipo de tareas dura el tiempo que se indica en la restricción de uso del escenario, y la fecha y hora en que debe ser iniciada esta tarea ya es un valor dado.

  Las restricciones de horario se transforman en restricciones de día calendario, generando una nueva restricción para cada día calendario en el horario que se indica en la restricción original.

  Las restricciones de día de la semana también se transforma a restricciones de día calendario, pero esta vez solo se añade una restricción de día calendario, para cada día calendario que cumple con tener el mismo día de la semana que la restricción de día de la semana a transformar.

[4] la restricción dos indica que los partidos de diferentes fechas y rondas se juegan en diferentes días calendario
[5] problema de organización de tareas en el tiempo.

Algunas restricciones de uso de escenario se solapan en el tiempo. En este caso, la implementación resuelve mezclando las restricciones de uso de escenario implicadas en el solapamiento, en una sola restricción que abarque el tiempo de todas la tareas solapadas e inicie el día calendario y la hora de la restricción con menor fecha-hora.

- La restricción 10, es acerca de restringir que si un arbitro juzga en 2 partidos diferentes, entonces estos partidos no se solapan en el tiempo.

  Para implementar esta restricción también se aplica una reducción de este problema en términos de $cumulatives$.

  Sin embargo la reducción se aplica un poco diferente. Para empezar, se lanza un propagador $cumulatives$ por cada árbitro del torneo. Cada vez que se lanza el propagador, al igual que antes, cada partido es una tarea que tarda un tiempo dp en ser terminada. Pero en este caso las máquinas disponibles son la máquina 0 y la máquina 1. La máquina 0 se define como una máquina que tiene recursos para atender todas las tareas del problema al mismo tiempo. La máquina 0 entonces representa que el árbitro para el cuál se está lanzando el propagador, no está juzgando este partido. La máquina 1, que representa el árbitro mencionado juzgando el partido, solo tiene recursos para atender una sola tarea por vez. Debido a que el objetivo del generador de soluciones es maximizar la función objetivo, se obtiene que el generador de soluciones tratará de asignar la mayor cantidad de árbitros posibles a todos los partidos.

- La restricción 11, acerca de las restricciones de uso de los árbitros, son tratadas como tareas que resuelven los árbitros, teniendo en cuenta que primero se transforman todas las restricciones de uso en restricciones de uso de tipo día calendario, y además mezclando en una sola las restricciones que se solapen.

9.4.2. Función Objetivo

La implementación de la función objetivo tiene dos partes importantes: 1) la asignación de variables auxiliares de decisión relacionadas con la función objetivo y 2) la implementación de la maximización de la función objetivo.

Las variables relacionadas con la función objetivo son: fo, cAE, cHR, $cAPnd$ y $cESCnd$. La asignación de estas variables se hace a través de la aplicación de propagadores que ayudan a mantener las relaciones matemáticas que se declararon en la sección 9.2.4.

La implementación de la máximización de la función objetivo se realiza por medio de la extensión de la clase IntMaximizeScript, la cual entre otras cosas, cuando es utilizada en conjunto con el algoritmo de búsqueda Branch&Bound, máximiza la variable de dominio finito que retorne la función $cost$. En nuestro caso se redefine $cost$ para retornar fo. Entonces cada vez que se encuentra una solución b, se adiciona la siguiente restricción en los siguientes espacios de búsqueda:

$fo > cost(b)$

9.4.3. Distribuidor

Las estrategias de distribución mencionadas en la sección 9.3 se configuran en el llamado a la función $branch$ de GECODE.

La clase de la cuál extendemos funcionalidad, IntMaximizeScript, ayuda a recibir parámetros enviados en la linea de comando, para cambiar la estrategia de distribución a realizar. El parámetro que indica cual estrategia elegir es $-branching$ y los posibles valores son $naive\_1$, $naive\_2$ y $on\_problem\_1$.

En el archivo Makefile puede encontrar ejemplos de como cambiar la estrategia de distribución y de cómo usar los diferentes parámestros que recibe la clase IntMaximizeScript. Para una mayor descripción de los diferentes parámetros que recibe esta clase y sobre el cómo cambiar la estrategia de búsqueda, puede consultar [13].

Capítulo 10

PRUEBAS

10.1. Introducción

El objetivo de las pruebas en este trabajo es identificar cuál de las distribuciones propuestas se ajusta mejor a la búsqueda de soluciones de nuestro problema, la organización de torneos de fútbol.

Explorar el comportamiento de la complejidad de los problemas en función de los parámetros de entrada sería interesante para tener un mejor entendimiento del problema y así diseñar mejores heurísticas de búsqueda y estrategias de distribución, sin embargo para lograr esto es necesario ejecutar de forma exhaustiva diferentes tipos de pruebas, lo cual queda por fuera del alcance del presente trabajo.

10.2. Generación de casos de prueba

La generación de casos de prueba implica abstraer un poco los parámetros de entrada, descritos en la sección 9.2.1, para identificar qué hace diferente un problema de otro.

En la lista que se muestra a continuación se describen las variables de caso de prueba que abstraen los parámetros mencionados, de tal forma que el cambio en una de estas variables implica que se está cambiando el tipo de problema, creando un caso de prueba diferente.

- n_equipos: número de equipo del torneo.
- tipo_torneo: tipo de torneo.
- n_horas_x_dia: número de horas disponibles por día, para programar los partidos.
- duaracion_partido: duración de cada partido.
- max_p_dia: máximo número de partidos que se va a permitir jugar durante un día, para el torneo.
- max_p_semana: máximo número de partidos que se va a permitir jugar durante una semana.
- n_dias: número total de días disponibles para el torneo.

- n_arbitro_x_partido: número de árbitros que se deben asignar a cada partido.
- n_arbitros: cantidad de árbitros disponibles para el torneo.
- experiencia_arbitros: arreglo de n_arbitro posiciones que indica la experiencia de cada árbitro.
- t_arbitro_restringido: arreglo de n_arbitro posiciones que indica el número horas en total que cada árbitro no va estar disponible para el torneo.
- n_escenearios: cantidad de escenarios disponibles para el torneo.
- c_iluminacion_escenario: arreglo de n_escenearios posiciones que indica para cada escenario si está o no íluminado.
- t_escenario_restringido: arreglo de n_escenearios posiciones que indica el número de horas total que cada escenario no va a estar disponible para el torneo.
- n_horas_horario_recomendado: número de horas que se van a recomendar de horario para el torneo.
- n_horas_dia_semana_recomendado: número de horas que se van a recomendar de día de la semana.
- n_horas_dia_calendario_recomendado: número de horas que se van a recomendar de día calendario.
- estrategia_de_distribucion: estrategia de distribución que se va a usar para buscar una solución.

Observando la anterior lista podemos identificar 18 puntos de bifurcación para la generación de nuevos tipos de casos de prueba, lo cual hace imposible explorarlos exhaustivamente a todos. Por ejemplo, si se simplificara cada punto de bifurcación de tal modo que se tuviera como máximo 2 posibles valores, se tendría un total de $2^{18} = 262144$ casos de prueba, para los cuales, si decidiéramos buscar solución durante máximo 10 minutos, necesitaríamos en el peor de los casos alrededor de 1820 días para explorarlos, es decir unos 5 años.

La implementación del script de generación de casos de prueba, entonces requiere tanto de la reducción de los posibles valores en cada punto de bifurcación, así como de un enfoque de generación de casos de forma no exhaustiva. En la sección siguiente se hará una pequeña discusión de las decisiones tomadas para la implementación del script generador de casos de prueba.

10.3. Implementación

La siguiente estructura de archivos muestra dónde están ubicados los scripts necesarios para generar los casos de prueba, calcular su solución y exportar los resultados.

```
- copista.com
    - modelo
        - management
            - commands
                - generar_casos_prueba.py
                - calcular_casos_prueba.py
                - exportar_casos_prueba.py
```

Para ejecutar cada uno de estos scripts se hace mediante el llamado de comandos de Django:

```
$ python manage.py generar_casos_prueba -n 10
$ python manage.py calcular_casos_prueba
$ python manage.py exportar_casos_prueba > results.csv
```

El ejemplo anterior crearía 10 tipos de casos de prueba diferente en la base de datos del sistema, luego se procedería al calculo de las soluciones y finalmente se exportarían los resultados a formato csv[1] para su posterior análisis.

Los scripts para calcular las soluciones de los casos de prueba y exportar los resultados son relativamente triviales, por lo cual se recomienda revisar el código en mencionados scripts para ampliar la información.

En el script de generación de casos se usa un enfoque estocástico para generar los distintos tipos de problemas que se quieren analizar, es decir, a cada variable de caso de prueba se le asigna un valor aleatorio de su dominio de posibles valores cada vez que se quiere generar un nuevo caso de prueba. Las tablas 10.1 y 10.2 resume las decisiones tomadas respecto a la reducción de dominios de las variables.

Adicionalmente el script de generación de casos permite indicar con cuáles estrategias de distribución se quiere buscar la solución para cada caso de prueba generado. Esta operación se puede realizar adicionando al llamado del script el parámetro $-d$, seguido por una lista separada de espacios indicando cuáles estrategias de distribución usar. Por ejemplo, si se llamara el script para crear un caso de prueba A aleatorio, y se adiciona el parámetro $-d$ seguido por la lista 0 1 2, se crearía el mismo caso A tres veces, pero indicando que para uno de los casos buscar la solución con la estrategia de distribución 0, el otro caso con la estrategia 1 y otro el último caso con la estrategia 2. La estrategia de distribución 0 se refiere a naive_1, la estrategia de distribución 1 es naive_2 y la estrategia de distribución 2 se refiere a on_problem_1.

Para generar los casos de prueba que se van a analizar durante este trabajo, se ejecutaron los siguientes comandos:

```
$ python manage.py generar_casos_prueba -n 144 -d 0 1 2
$ python manage.py calcular_casos_prueba
$ python manage.py exportar_casos_prueba > results.csv
```

Lo cual creó 144 tipos diferentes de casos de prueba para cada una de las estrategias de distribución disponible, luego se calculó las soluciones para cada caso y finalmente se creó el archivo results.csv, el cuál contiene los datos necesarios para hacer el análisis de resultados.

10.4. Resultados

Las gráficas mostradas en las figuras 10.1 a 10.7 son generadas a partir de los datos en el archivo results.csv. Se debe tener en cuenta que para generar las gráficas se quitaron los casos en los que no se logró generar solución, y que para encontrar la mejor solución posible en cada caso de prueba se dio un tiempo de espera máximo de 10 minutos.

[1] archivo de valores separados por comas, del inglés comma-separated values.

Variable	Dominio	Descripción
n_equipos	{4, 8, 16}	Se reduce el dominio a estos 3 posibles valores, los cuales son potencia de 2, facilitando así los cálculos de cantidad de partidos del torneo y limitando cuán grande es un problema.
tipo_torneo	{0, 1}	Estos son los únicos posibles valores de esta variable. 0 significa un torneo tipo todos contra todos y 1 significa que es un torneo por eliminatorias
n_horas_x_dia	{4, 8, 12, 16}	Se escogen estos posibles valores, lo cuales se consideran los más habituales.
duaracion_partido	{2}	Se estima que este es el valor más habitual de la duración de un partido de fútbol, teniendo en cuenta los 90 minutos reglamentario más el tiempo complementario y el tiempo que se tarda en organizar las personas en la tribuna de juego.
max_p_dia	{1, ..., valor máximo}	El valor máximo del dominio de esta variable es un valor límite desde el cual la variable deja de limitar las posibles soluciones del problema. Este valor límite se cálcula teniendo en cuenta n_horas_x_dia, duaracion_partido, n_escenearios y la máxima cantidad de pártidos por fecha.
max_p_semana	{1, ..., valor máximo}	El valor máximo es igual a $7 \times max\_p\_dia$. Al igual que en la anterior variable, valores por encima del valor máximo no restringen las soluciones del problema.
n_dias	{valor mínimo, ..., 120 % del valor mínimo}	Se cálcula un valor mínimo de días disponibles para el torneo para que el problema tenga una solución fáctible. Este valor se cálcula teniendo en cuenta el número de equipos, el tipo de torneo, max_p_dia y max_p_semana. El límite superior del dominio es árbitrario y se define así para limitar la cantidad de tipos diferentes de casos.
n_arbitro_x_partido	{3}	Debido a que el ámbito de uso del sistema son federaciones, clubes y demás entidades a nivel regional, lo más probable es que se definan 3 árbitros por partido, por ello solo se estudian problemas con este valor.
n_arbitros	{3, ..., valor máximo }	El valor máximo es igual a $n\_arbitro\_x\_partido \times max\_p\_dia$. Es decir que se estudiaran casos en los que se tengan cómo mínimo la cantidad suficiente de árbitros para abastecer un partido, hasta casos donde haya suficientes árbitros como para asignar diferentes árbitros a todos los partidos de un día.

Cuadro 10.1: Variables y dominios de los casos de prueba.

En la figura 10.1 se puede observar que la estrategia de distribución naive_1 en promedio genera las peores soluciones, mientras que la estrategia de distribución on_problem_1 es en promedio la que genera mejores soluciones. El valor negativo de la función objetivo en la estrategia de distribución naive_1 se puede interpretar como que la estrategia encontró soluciones en las que no se asignaban ni escenarios ni

Variable	Dominio	Descripción
experiencia _arbitros	$\{EA_1, EA_2, ..., EA_n\}$ $EA_i \in \{1, ..., 10\}$	Para cada tipo de problema esta variable es un arreglo de $n = n\_arbitros$ elementos, representando la experiencia de cada árbitro, donde el dominio de cada experiencia es un valor entre 1 y 10.
t_arbitro _restringido	$\{RA_1, RA_2, ..., RA_n\}$ $RA_i \in \{0, ..., \frac{T}{3}\}$	Igual que en la anterior variable, se indica la cantidad de horas restringidas para cada árbitro. T significa la cantidad total horas disponibles del torneo.
n_escenearios	{1, ..., n_equipos }	Se define que cómo máximo hay n_equipos canchas de fútbol por partido.
c_iluminacion _escenario	$\{IE_1, IE_2, ..., IE_n\}$ $IE_i = 1$	Se indica que todos los escenarios están íluminados, para no agregar restricciones de horas en las que los escenarios están disponibles, aportando así simplicidad al análisis de los resultados.
t_escenario _restringido	$\{RE_1, RE_2, ..., RE_n\}$ $RE_i \in \{0, ..., \frac{T}{3}\}$	Igual que para los árbitros, se indica la cantidad de horas restringidas para cada escenario. T significa la cantidad total horas disponibles del torneo.
n_horas _horario _recomendado	$\{0, ..., \frac{n\_horas\_x\_dia}{3}\}$	Se asigna el dominio teniendo en cuenta que no se recomiende más de la tercera parte de las horas de un mismo día.
n_horas _dia_semana _recomendado	$\{0, ..., \frac{n\_horas\_x\_dia \times 7}{3}\}$	Igual que la anterior variable, pero en este caso teniendo en cuenta el total de horas disponibles en la semana.
n_horas _dia_calendario _recomendado	$\{0, ..., \frac{T}{3}\}$	Igual que las anteriores variables, pero teniendo en cuenta el total de horas disponibles de todo el torneo, T.
estrategia_de _distribucion	{0, 1, 2}	0 representa la estrategia de distribución naive_1, 1 representa naive_2 y 2 representa on_problem_1.

Cuadro 10.2: Variables y dominios de los casos de prueba.

árbitros a los partidos[2].

Las figuras 10.2 y 10.3 muestra nuevamente el promedio de los valores de función objetivo (FO), pero analizando su comportamiento con diferente número de equipos. La figura 10.2 analiza los casos con torneos todos contra todos, mientras que la figura 10.3 analiza los casos con torneos por eliminatorias. En ambas gráficas se puede observar de nuevo que la estrategia de distribución on_problem_1 es la que genera mejores soluciones. Adicionalmente, en la figura 10.2 se puede observar que la estrategia naive_1 fue la única capaz de generar soluciones para torneos con 16 equipos, todos contra todos.

Dado el número de equipos y el tipo de torneo se puede calcular cuantos partidos se jugarán en el torneo. La figura 10.4 analiza los valores promedio de FO en función del número de partidos, lo cual nos da una mejor idea de cuan complejo es un caso de prueba. Nuevamente vemos un mejor rendimiento[3] con la estrategia de distribución on_problem_1. En general, respecto a las tres soluciones, se puede observar que el valor absoluto de la función objetivo tiende a ser proporcional a la cantidad de partidos del torneo[4].

[2]recuerde que el no asignar árbitros y escenarios a los partidos tiene un peso negativo significativamente alto en la función objetivo

[3]mejor rendimiento entendido en este caso como la estrategia que genera mejores soluciones.

[4]Tenga en cuenta que aunque parece haber comportamientos de orden exponencial, en realidad esto se debe a que los valores en el eje x no están a una escala lineal.

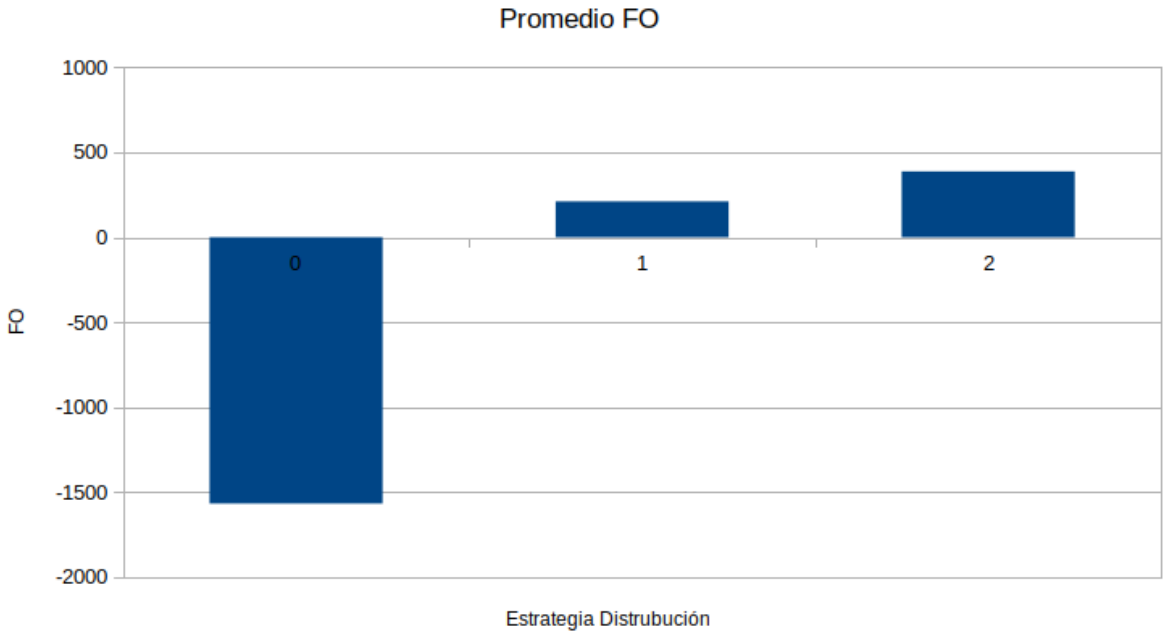

Figura 10.1: Comparación entre el promedio FO por cada estrategia de distribución.

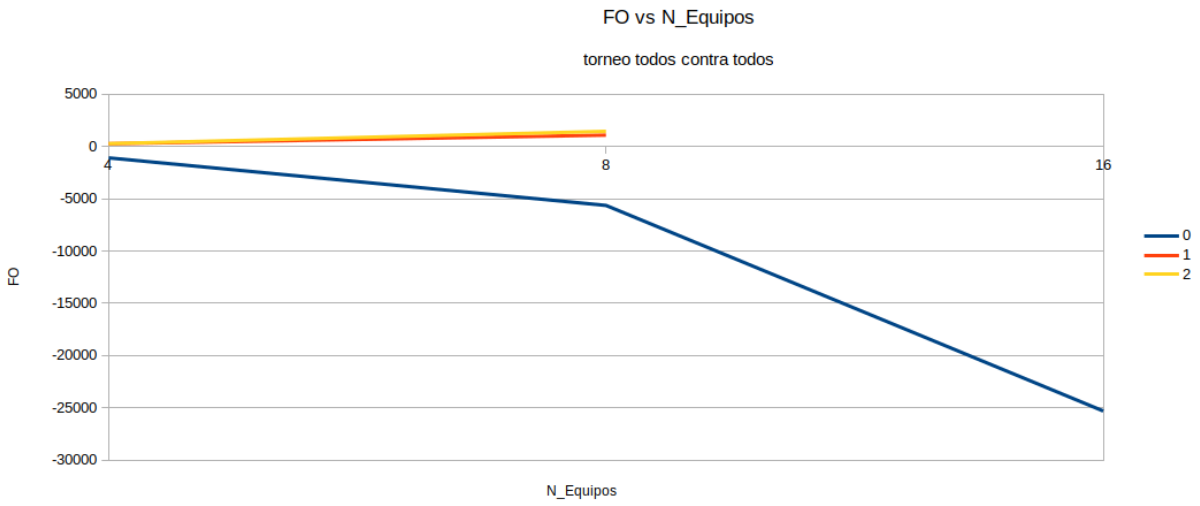

Figura 10.2: Comparación entre el promedio FO por cada estrategia de distribución. En este caso se analiza respecto al número de equipos en los torneos todos contra todos.

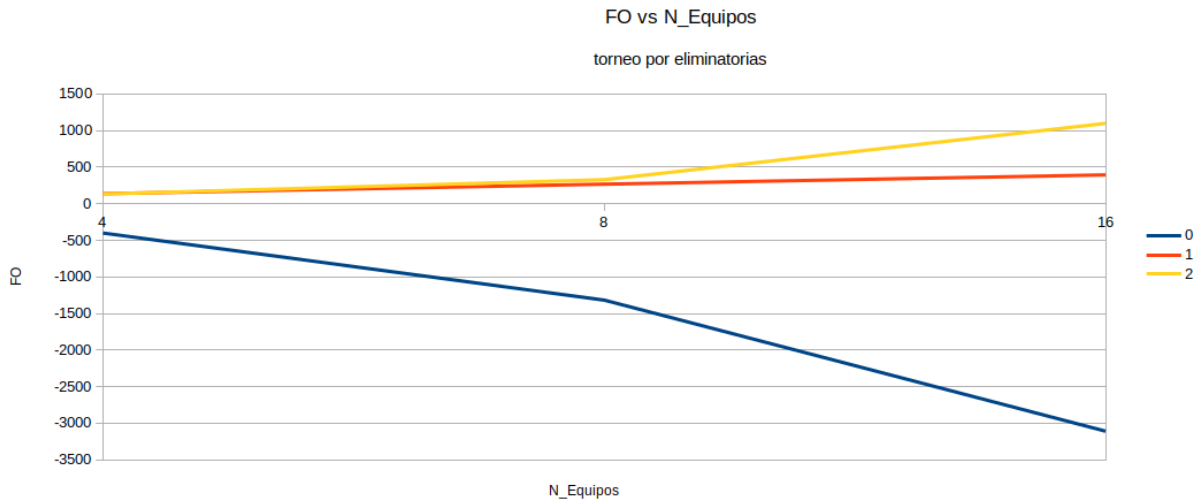

Figura 10.3: Comparación entre el promedio FO por cada estrategia de distribución. En este caso se analiza respecto al número de equipos en los torneos por eliminatorias.

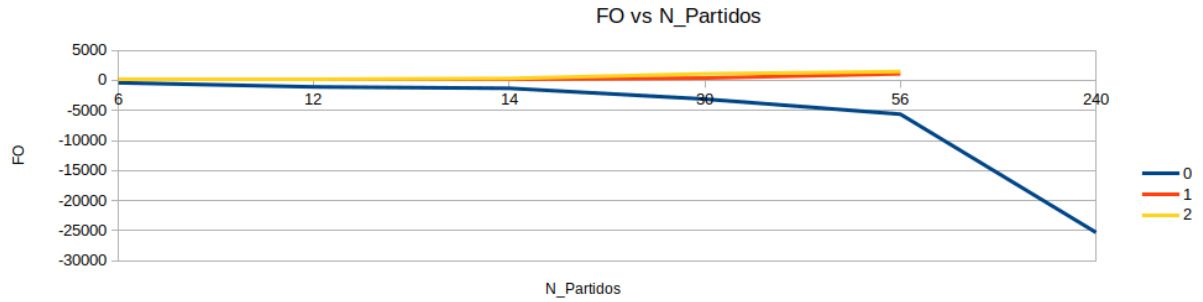

Figura 10.4: Comparación entre el promedio FO por cada estrategia de distribución. En este caso se analiza respecto al número de partidos.

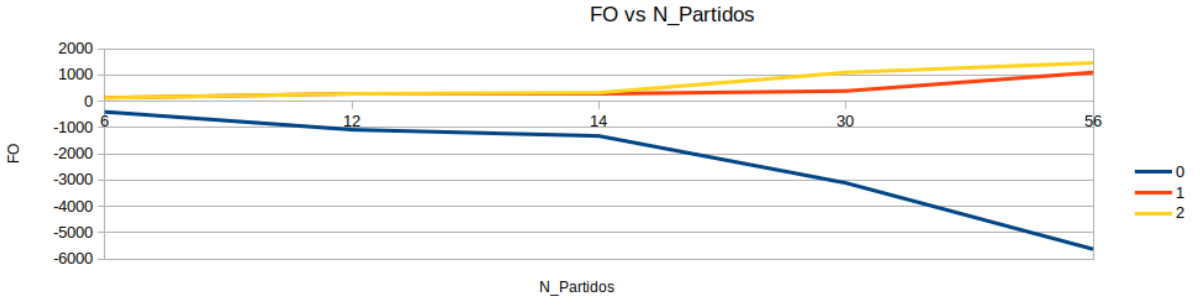

Figura 10.5: Misma gráfica que en la figura 10.4, pero analizando solo hasta los 56 partidos.

La figura 10.5 muestra de nuevo la misma información que la figura 10.4, pero dejando de lado los torneos con 240 partidos. En adelante se evitara realizar la comparación con los torneos que tenían 240, puesto que solo la estrategia naive_1 es capaz de obtener soluciones para estos casos y estas soluciones no son deseables dado a que no se están asignando escenarios y/o árbitros.

Un análisis de como se asignan los escenarios y los árbitros en las diferentes estrategias de distribución, se puede encontrar en las figuras 10.6 y 10.7. Tal y como se había supuesto antes, el problema de la

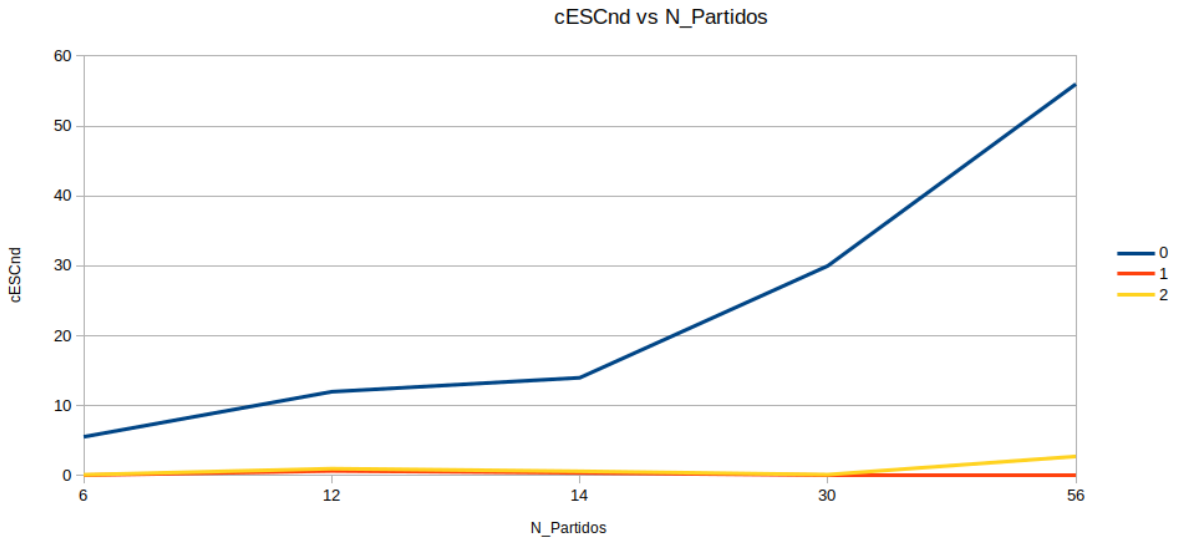

Figura 10.6: Comparación entre el promedio cESCnd contra número de partidos, para cada estrategia de distribución.

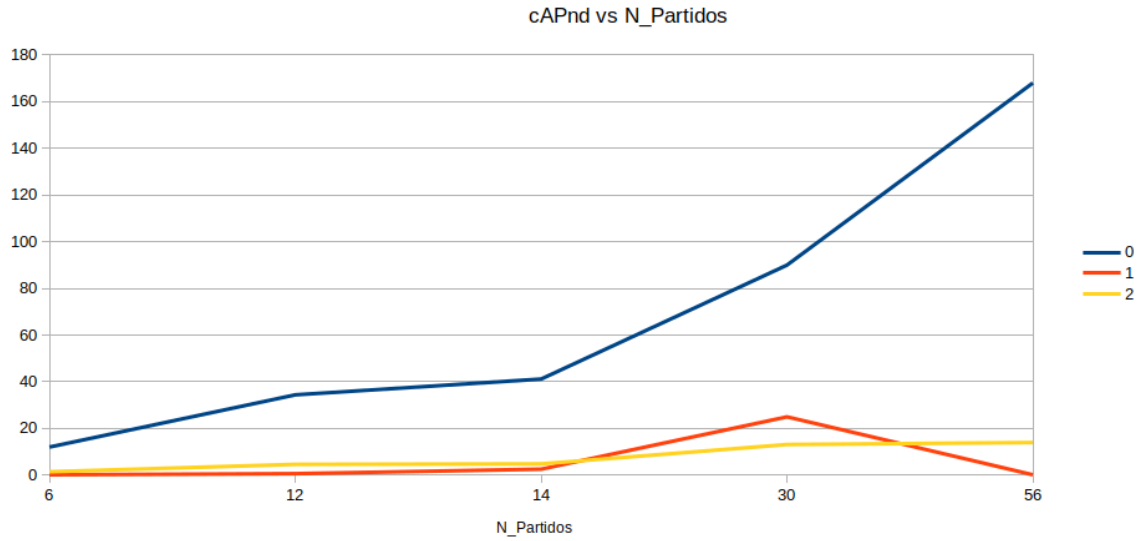

Figura 10.7: Comparación entre el promedio cAPnd contra número de partidos, para cada estrategia de distribución.

estrategia naive_1 es que no está asignando ni árbitros ni escenarios. Las otras 2 estrategias de distribución tienen un comportamiento similar en este aspecto. Las figuras 10.8 y 10.9 muestran la misma relación, pero esta vez en el eje y tenemos el porcentaje de los escenarios y árbitros no asignados. Estás últimas figuras hacen más evidente la tendencia a no asignar los escenarios y árbitros de los partidos, por parte de la estrategia naive_1.

Un análisis sobre la variable cHR, figura 10.10, nos permite observar que las 3 estrategias de distribución tienen un compartimiento promedio similar. Se podría decir que la estrategia on_problem_1 tiende a ser la mejor y que la estrategia naive_2 tiende a ser la peor en este aspecto.

Respecto a la variable cAP, figura 10.11: la estrategia de distribución naive_1 no busca soluciones que optimicen esta variable, mientras que on_problem_1 (siendo esta la que mejor optimiza) y naive_2 sí.

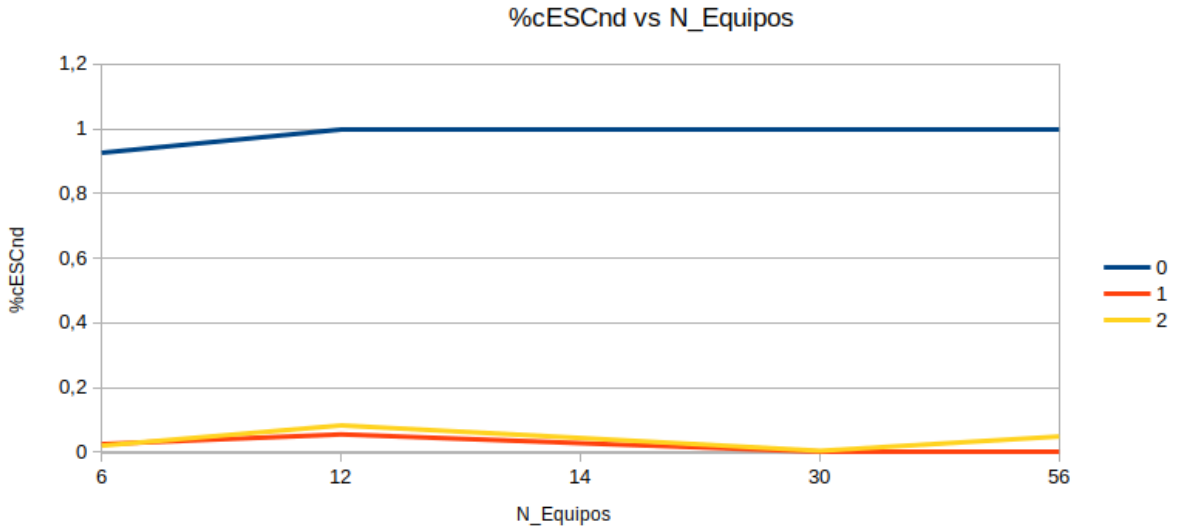

Figura 10.8: Comparación entre el promedio del porcentaje de escenarios no asignados contra el número de partidos.

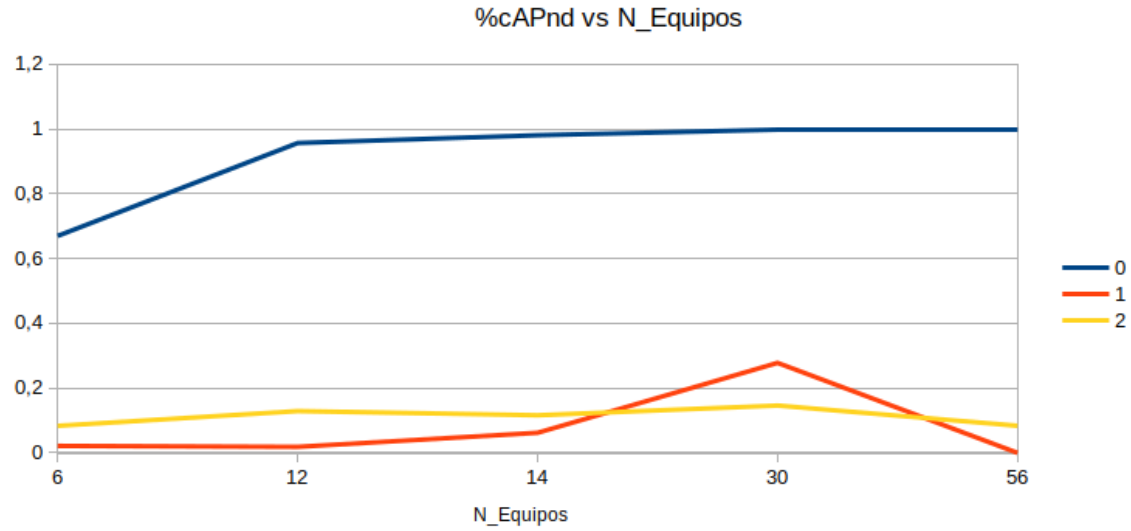

Figura 10.9: Comparación entre el promedio del porcentaje de árbitros no asignados contra el número de partidos.

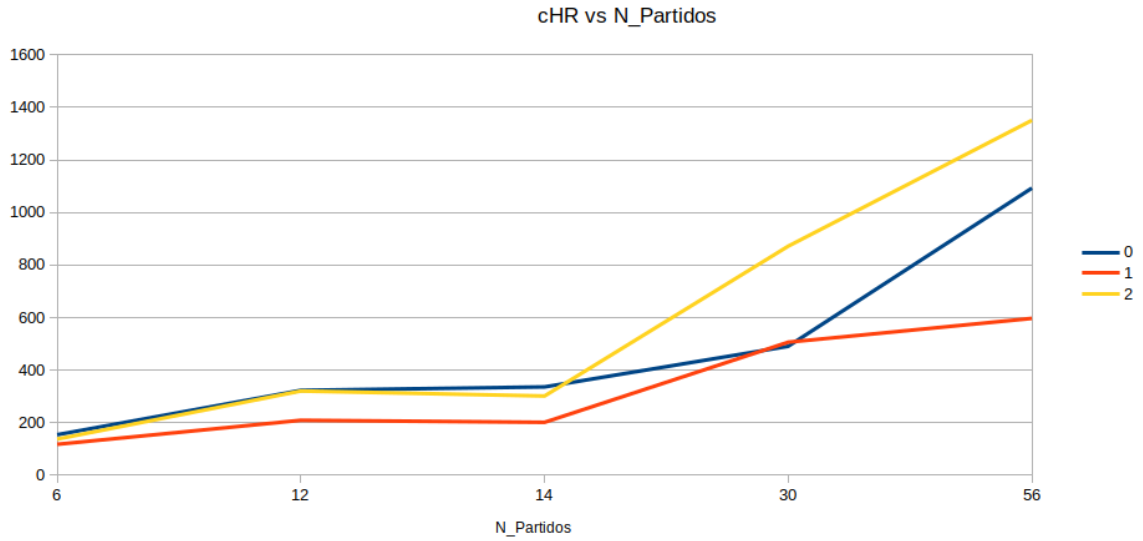

Figura 10.10: Comparación entre el promedio cHR contra número de partidos, para cada estrategia de distribución.

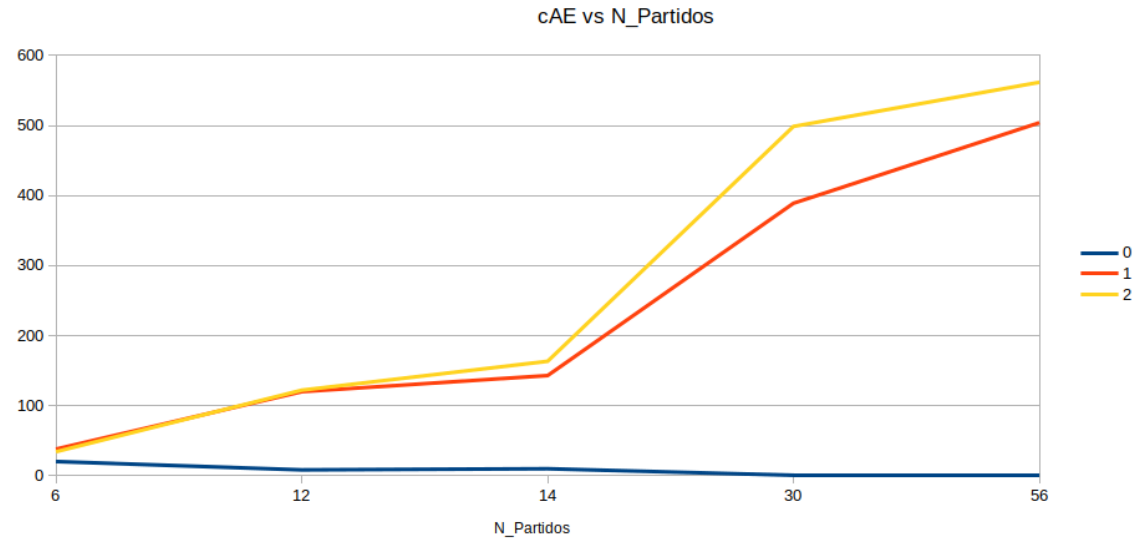

Figura 10.11: Comparación entre el promedio cAE contra número de partidos, para cada estrategia de distribución.

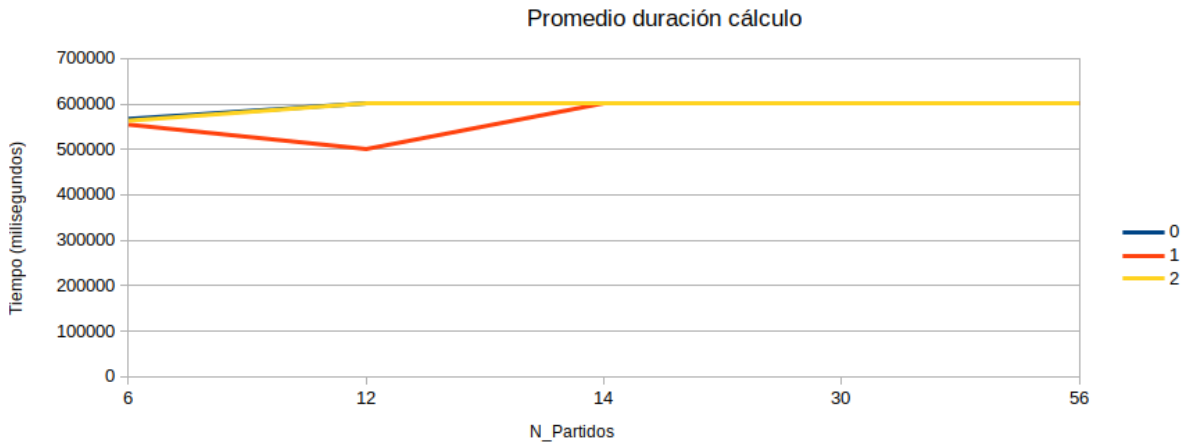

Figura 10.12: Comparación entre el promedio del tiempo en milisegundos que se tardo en encontrar las soluciones.

Finalmente, de la figura 10.12 se puede concluir que, en promedio, el cálculo de la solución para cada caso de prueba se tomo los 10 minutos de espera máximos, configurados previamente.

Capítulo 11

CONCLUSIONES Y TRABAJO FUTURO

De acuerdo a los resultados de las pruebas se puede concluir que el sistema puede ser usado para buscar soluciones al problema de organización de torneos en un tiempo relativamente admisible para casos con menos de 240 partidos. Además, de las posibles configuraciones exploradas en el sistema, seleccionar on_problem_1 como estrategia de distribución es la mejor opción, puesto que entre las disponibles es la que genera mejores soluciones.

El sistema aún puede ser mejorado tanto a nivel de sistema de información como a nivel de optimización de las estrategias de generación de soluciones de los problemas. Por ello se estima la siguiente lista de actividades como trabajo futuro:

- Realización de pruebas más exhaustivas que permitan realizar un análisis de datos que nos permitan tener un mejor entendimiento de cómo los parámetros de entrada afectan la complejidad del problema. De este modo se podría diseñar mejores heurísticas de búsqueda y estrategias distribución que ayuden a encontrar mejores soluciones.

- Búsqueda de cómo introducir restricciones redundantes, que aceleren el proceso de reducción de dominios de las variables, lo cual de ese modo acelera el proceso de búsqueda de mejores soluciones. Un ejemplo de esto sería lograr implementar la restricción 3, que aunque es redundante, aporta al mejoramiento del rendimiento de la aplicación en este sentido.

- Otra técnica en la programación por restricciones, que quedó por fuera del alcance de este proyecto y que ayuda mucho a acelerar el proceso de búsqueda de soluciones, es la ruptura de simetrías, es decir descartar soluciones que son simétricamente equivalentes, para así disminuir el árbol de búsqueda. Un ejemplo podría ser la asignación de árbitros en los partidos, una solución que asigne los árbitros A, B y C a un partido, es simétricamente equivalente a una solución que asigne los árbitros B, A y C al mismo partido.

- También se podría experimentar con el uso de metaheurísticas que ayuden a guiar el proceso de distribución de variables de tal modo que se reduzcan los costos computacionales de optimizar una solución. Por ejemplo, se podría experimentar con optimizaciones locales por partido, explorando de este modo si este enfoque mejora el rendimiento computacional sin sacrificar optimizaciones globales. Otro caso podría ser la inclusión de algoritmos estocásticos, aprendizaje de máquina o la combinación de ellos paralelo, para descubrir como guiar la búsqueda de soluciones de forma más eficiente.

- El sistema también se podría extender para que se adecue más a la realidad de la organización de torneos de fútbol. Como ejemplos: 1) se podría extender el sistema para que organice más de un torneo por vez, 2) se podría adicionar políticas para que en los torneos que admiten rondas de ida y vuelta, se generen partidos en los que los equipos tengan la oportunidad de jugar una vez como equipo local y otra como visitante en cada fecha del torneo, y 3) adicionar restricciones que tengan en cuenta los desplazamientos de los árbitros entre los diferentes escenarios.

    - Apoyando esta idea también se podría investigar cómo emplear la optimización estocástica[1] para representar de una forma más fiel a la realidad nuestro problema. Ejemplos de variables no deterministas pueden ser el tiempo en que tarda un árbitro de ir entre un escenario y otro, y la duración de los partidos, la cual depende de si hay tiempos extras o penales por empates, o si hay alargues en el tiempo de juego.

- A nivel de sistema de información, el servicio web creado bajo este trabajo requiere ser mejorado para permitir ser publicado en producción. Entre las diferentes mejoras posibles se podría mencionar: 1) adicionar un sistema de seguridad para que solo usuarios registrados pudieran usar el servicio y 2) adicionar una estrategia para evitar tener corriendo en paralelo el cálculo de soluciones de diferentes problemas pertenecientes a diferentes usuarios, esto podría hacer colapsar el servidor.

[1] Puede encontrar una introducción a este enfoque en [30].

Apéndice A

ESPECIFICACIÓN DE REQUERIMIENTOS

A.1. Requerimientos Funcionales

ID	RF-1
Nombre	Sortear equipos
Descripción	El sistema debe asignar aleatoriamente la organización de los equipos de cada torneo indicando quién es el equipo 1, 2, 3 y así sucesivamente, para indicar cómo van a ser los enfrentamientos de la primer ronda de cada torneo. Los equipos que pertenecen a la misma localidad del torneo deben ocupar los últimos puestos de la organización, garantizando de este modo, en el caso de que haya un número de equipos impares, que los equipos locales pasen automáticamente a la siguiente ronda.
Entrada	Localidad del torneo, lista de los equipos con nombre y localidad.
Salida	Posición que le corresponde a cada equipo.

ID	RF-2
Nombre	Seleccionar fechas y horarios de los partidos
Descripción	El sistema debe seleccionar las fechas y horas de los partidos de cada torneo. Para esto se debe tener en cuenta lo siguiente: 1. Respetar que en cada torneo se juegue un número menor de partidos por semana que el indicado como máximo en cada torneo. 2. No se debe jugar más partidos por día que el número máximo de partidos por día indicado en la entrada. 3. Tener en cuenta que la fecha del último partido de cada torneo debe ser antes de la fecha de finalización de dicho torneo. 4. Tener en cuenta que la fecha del primer partido de cada torneo debe ser después de la fecha inicio de dicho torneo. 5. No se puede jugar dos partidos en el mismo escenario en la misma fecha y hora. 6. Un equipo no puede jugar más de un partido por día. 7. Hacer que la mayoría de los partidos se jueguen en los horarios, días y fechas recomendadas, dando más peso a los torneos con mayor prioridad.
Entrada	Fecha de inicio y fin del torneo, número de equipos que participan en el torneo, prioridad del torneo, máximo de partido por semana, máximo de partidos por día, horario (día o noche) y día de la semana recomendados, fecha (dia, mes, año) y horario (día o noche) recomendados.
Salida	Fecha y horarios de los partidos del torneo.

ID	RF-3
Nombre	Seleccionar escenarios para cada partido
Descripción	El sistema debe seleccionar los escenarios de los partidos de cada torneo. Para esto se debe tener en cuenta lo siguiente: 1. Asignar escenarios a los partidos, respetando las restricciones de escenarios, es decir las fechas y horarios en los que cada escenario no estarán disponibles. 2. Tener en cuenta que no se puede jugar partidos en las noches, en escenarios que no tienen iluminación. 3. No se puede jugar dos partidos en el mismo escenario en la misma fecha y horario.
Entrada	Escenarios disponibles, restricciones de uso del escenario, indicar que escenarios cuentan con iluminación.
Salida	Escenario donde se jugará cada partido del torneo.

ID	RF-4
Nombre	Seleccionar jueces de los partidos
Descripción	El sistema debe seleccionar los jueces de los partidos de cada torneo. Para esto se debe tener en cuenta lo siguiente: 1. Para asignar los jueces, se debe tener en cuenta que el juez no debe juzgar más de un juego en el mismo día y horario. 2. Se debe tratar que la mayoría de partidos de las finales y semi-finales sean juzgados por los jueces con mayor experiencia.
Entrada	Jueces disponibles y experiencia de cada uno.
Salida	Juez que juzgará cada partido del torneo.

A.2. Requerimientos No Funcionales

La siguiente es una lista de requisitos no funcionales del sistema. El esquema de clasificación de requisitos implementado, es el propuesto en [31].

ID	RNF-1
Categoría	Facilidad de Uso
Descripción	Con el fin de disminuir la desorientación entre los usuarios del servicio web, el sistema debe proveer una interfaz del tipo Restful API, de modo que armonice con las implementaciones realizadas en el sistema SIND. Esta Restful API debe ser presentada con en formato JSON y debe permitir recibir la siguiente información de los requerimientos RF-1 al 4: ▪ Localidad del torneo, lista de los equipos con nombre y localidad. ▪ Fecha de inicio y fin del torneo, número de equipos que participan en el torneo, prioridad del torneo, máximo de partido por semana, máximo de partidos por día, horario (día o noche) y día de la semana recomendados, fecha (dia, mes, año) y horario (día o noche) recomendados. ▪ Escenario se jugará cada partido del torneo. ▪ Jueces disponibles y experiencia de cada uno. De igual forma, la solución debe mostrar la información que propone como salida en los requerimientos RF-1 al 4: ▪ Posición que le corresponde a cada equipo. ▪ Fecha y horarios de los partidos del torneo. ▪ Escenario se jugará cada partido del torneo. ▪ Juez que juzgará cada partido del torneo.

ID	RNF-2
Categoría	Soporte
Descripción	La adición o eliminación de requerimientos respecto a las restricciones a tener en cuenta en la organización de torneos de fútbol, están supeditados a los cambios en las políticas y estrategias internas por parte de las entidades organizadoras de estos torneos. Por esto el sistema debe ser cuidadosamente diseñado y estructurado, de tal forma que en futuras fases de mantenimiento y desarrollo del sistema, puedan ser añadidas fácilmente.

Bibliografía

[1] PUBLICACIONES SEMANA S.A. Fútbol en colombia: pasión e identidad. `http://www.semana.com/nacion/articulo/futbol-en-colombia-pasion-identidad/384019-3`, April 2014. Accessed: 2016-06-09.

[2] Irvin J. Lustig and Jean-François Puget. Program does not equal program: Constraint programming and its relationship to mathematical programming. *Interfaces*, 31(6):29–53, November 2001.

[3] Jaime Leiva and Mauricio Gaona. *Diseño del Sistema de Información del Deporte, la Recreación y la Actividad Física*. Proyecto colciencias, Universidad del Valle, 2015.

[4] Graham Kendall, Sigrid Knust, Celso C. Ribeiro, and Sebastián Urrutia. Scheduling in sports: An annotated bibliography. *Computers & Operations Research*, 37(1):1 – 19, 2010.

[5] Mike DiNunzio and Serge Kruk. Soccer tournament scheduling using constraint programming. In *Proceedings of the 8th International Conference on the Practice and Theory of Automated Timetabling (PATAT 2010)*, pages 193–200, 2010.

[6] Thomas Bartsch, Andreas Drexl, and Stefan Kröger. Scheduling the professional soccer leagues of austria and germany. *Computers & Operations Research*, 33(7):1907–1937, July 2006.

[7] F. Della Croce and D. Oliveri. Scheduling the italian football league: An ilp-based approach. *Computers & Operations Research*, 33(7):1963–1974, July 2006.

[8] Guillermo Durán, Mario Guajardo, Jaime Miranda, Denis Sauré, Sebastián Souyris, Andres Weintraub, and Rodrigo Wolf. Scheduling the chilean soccer league by integer programming. *Interfaces*, 37(6):539–552, November 2007.

[9] Celso C. Ribeiro and Sebastián Urrutia. *Scheduling the Brazilian Soccer Tournament with Fairness and Broadcast Objectives*, volume 3867 of *Lecture Notes in Computer Science*, pages 147–157. Springer Berlin Heidelberg, 2007.

[10] Luis Felipe Vargas Rojas and Juan Francisco Díaz Frias. Diseño e implementación de una aplicación web y móvil para consultar sabio. Trabajo de grado, Universidad del Valle, 2014.

[11] Roman Barták. Constraint programming: In pursuit of the holy grail. In *Proceedings of the Week of Doctoral Students (WDS99 -invited lecture*, pages 555–564, 1999.

[12] Krzysztof Apt. *Principles of Constraint Programming*. Cambridge University Press, New York, NY, USA, 2003.

[13] Gecode: Documentation. `http://www.gecode.org/documentation.html`. Accessed: 2017-08-14.

[14] Thomas H. Cormen, Clifford Stein, Ronald L. Rivest, and Charles E. Leiserson. *Introduction to Algorithms*. McGraw-Hill Higher Education, 2nd edition, 2001.

[15] W3C web service definition. `https://www.w3.org/TR/2004/NOTE-ws-gloss-20040211/#webservice`. Accessed: 2016-06-09.

[16] Relationship to the world wide web and rest architectures. `https://www.w3.org/TR/2004/NOTE-ws-arch-20040211/#relwwwrest`. Accessed: 2017-08-17.

[17] Roy Thomas Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, 2000. AAI9980887.

[18] Standard streams. `https://www.gnu.org/software/libc/manual/html_node/Standard-Streams.html`. Accessed: 2017-08-17.

[19] The mozart programming system. `http://mozart.github.io/`. Accessed: 2017-08-14.

[20] Django rest framework: Quickstart. `http://www.django-rest-framework.org/tutorial/quickstart/`. Accessed: 2017-08-14.

[21] Gecode: Interfaces. `http://www.gecode.org/interfaces.html`. Accessed: 2017-08-14.

[22] Gecode for python project. `https://code.launchpad.net/gecode-python`. Accessed: 2017-08-14.

[23] Thomas Erl. *Service-Oriented Architecture: Concepts, Technology, and Design*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2005.

[24] Thorsserializer. `https://github.com/Loki-Astari/ThorsSerializer`. Accessed: 2017-08-14.

[25] Sqlite home page. `https://www.sqlite.org/`. Accessed: 2017-08-14.

[26] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-oriented Software*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1995.

[27] Craig Larman. *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development (3rd Edition)*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2004.

[28] Nicolas Beldiceanu and Mats Carlsson. A new multi-resource cumulatives constraint with negative heights. In *Principles and Practice of Constraint Programming - CP 2002, 8th International Conference, CP 2002, Ithaca, NY, USA, September 9-13, 2002, Proceedings*, pages 63–79, 2002.

[29] Pascal Van Hentenryck, editor. *Principles and Practice of Constraint Programming - CP 2002, 8th International Conference, CP 2002, Ithaca, NY, USA, September 9-13, 2002, Proceedings*, volume 2470 of *Lecture Notes in Computer Science*. Springer, 2002.

[30] Lauren A. Hannah. Stochastic optimization, 2014.

[31] Robert B. Grady. *Practical Software Metrics for Project Management and Process Improvement*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1992.