PONTIFICIA UNIVERSIDAD CATÓLICA DE VALPARAÍSO
FACULTAD DE INGENIERÍA
ESCUELA DE INGENIERÍA INFORMÁTICA

MODELADO Y RESOLUCIÓN DEL NURSE ROSTERING PROBLEM (NRP) UTILIZANDO PROGRAMACIÓN CON RESTRICCIONES: UN CASO DE ESTUDIO

RENZO ANDRÉS PIZARRO HIDALGO

GIANNI CARLO RIVERA ROJAS

INFORME FINAL DEL PROYECTO
PARA OPTAR AL TÍTULO PROFESIONAL DE
INGENIERO DE EJECUCIÓN EN INFORMÁTICA

OCTUBRE 2011

PONTIFICIA UNIVERSIDAD CATÓLICA DE VALPARAÍSO
FACULTAD DE INGENIERÍA
ESCUELA DE INGENIERÍA INFORMÁTICA

MODELADO Y RESOLUCIÓN DEL NURSE ROSTERING PROBLEM (NRP) UTILIZANDO PROGRAMACIÓN CON RESTRICCIONES: UN CASO DE ESTUDIO

RENZO ANDRÉS PIZARRO HIDALGO

GIANNI CARLO RIVERA ROJAS

Profesor Guía: **Ricardo Soto De Giorgis**
Profesor Co-referente: **Broderick Crawford Labrín**

Carrera: **Ingeniería de Ejecución en Informática**

OCTUBRE 2011

Dedicatoria

A nuestras familias por ser el pilar emocional fundamental
en el largo camino del aprender.

Agradecimientos

A Dios, a nuestras familias por su incondicional apoyo en todo momento, a nuestro profesor guía por su paciencia y compromiso con nuestro trabajo, a Clínica Valparaíso que facilitó la información para llevar a cabo este proyecto y en general a todas las personas cercanas que nos brindaron afecto y buenos consejos en esta etapa de nuestras vidas.

Resumen

El presente proyecto tiene como objetivo el modelado y resolución del Nurse Rostering Problem (NRP) utilizando programación con restricciones. El NRP consiste en la asignación de turnos para enfermeras de un centro médico teniendo en consideración un conjunto de restricciones, tales como la cantidad y tipos de turnos, número de enfermeras por turno, número máximo y mínimo de horas por semana, entre otras.

En este documento se modela un problema de satisfacción de restricciones basado en la planificación de turnos utilizados por la Clínica Valparaíso. Se presenta un modelo inicial y otro avanzado del problema, los cuales son resueltos en ECL^{i}PS^e. Se comparan distintas estrategias de enumeración con el fin de encontrar la que presente mejores tiempos de resolución.

Palabras claves: Programación con Restricciones, Problema de Asignación de Enfermeras.

Abstract

This project aims at modeling and solving the Nurse Rostering Problem (NPR) by using constraint programming. The NRP consists in assigning working shifts to nurses, taking into account a set of constraints, such as the amount and types of shifts, number of nurses per shift, maximum and minimum number of hours per week, among others.

In this document, a constraint satisfaction problem is modeled based on the shift planning of the Clínica Valparaiso. We present an initial model and a more advanced one are, which are solved in ECL^{i}PS^e. We compare and analyze different enumeration strategies in order to conclude which performs better in terms of solving time.

Keywords: Constraint Programming, Nurse Rostering Problem.

Índice

Lista de Figuras

Lista de Tablas

1. Introducción

La programación con restricciones es un paradigma de programación principalmente dedicado a la resolución de problemas de satisfacción de restricciones, comúnmente conocidos como Constraint Satisfaction Problems (CSP) en inglés. Las áreas de aplicación de la programación con restricciones son diversas. El comienzo de ésta, nace con los primeros trabajos relacionados con la resolución de problemas en el ámbito de la inteligencia artificial. Con el pasar de los años se han incrementado los campos de aplicación, utilizándose en scheduling, planificación, logística y en la toma de decisiones.

Muchos de los problemas que se generan en los campos mencionados, son modelados como CSP, los cuales se representan mediante variables, dominios y restricciones. El objetivo de este proyecto es modelar y resolver el Nurse Rostering Problem (NRP), el cual consiste básicamente en la asignación de turnos para enfermeras de un centro médico, donde se tenga en cuenta cada una de las disposiciones y normas legales que pueden estar incluidas, que en definitiva serán modeladas como restricciones.

Para comprender de mejor manera el modelado, es necesario introducir ciertos contenidos importantes, que van desde la formalización matemática de un CSP hasta heurísticas de ordenamiento de variable y selección de valor. Asimismo, conocer diferentes tipos de *solvers* (motores de búsqueda) según su distinta naturaleza con el objetivo de escoger la opción más adecuada.

En cuanto al modelo, trabajar con la disposición y normas legales cercanas a la realidad puede llevar a la investigación por buen camino para cumplir objetivos, por lo que se dará especial énfasis en conseguir información real y certera. Particularmente el modelado del presente proyecto se fundamenta en el caso de estudio de Clínica Valparaíso, separado en un modelo preliminar que incluye el sistema real de trabajo con la normativa básica que ha sido extraída y un modelo avanzado, el cual incorporará un nuevo rol con sus respectivas restricciones.

Con la etapa previa de modelado elaborada, se pondrán a prueba variadas heurísticas con la finalidad de comparar y dilucidar el mejor comportamiento basándose en ciertos criterios de rendimiento.

2. Objetivos

Objetivo general

Modelar y resolver dos versiones del Nurse Rostering Problem (NRP) basadas en el problema de planificación de turnos de la Clínica Valparaíso, utilizando programación con restricciones.

Objetivos específicos

- Modelar el NRP de Clínica Valparaíso utilizando programación con restricciones
- Resolver versión preliminar del NRP utilizando ECL^{i}PS^e
- Resolver versión avanzada del NRP utilizando ECL^{i}PS^e
- Comparar y analizar los resultados del NRP avanzado utilizando distintas estrategias de enumeración, con el fin de determinar la mejor

3. Estado del arte

El NRP se aplica a la vida real para la asignación correcta de turnos de acuerdo a un conjunto de restricciones. Diversos autores se han dedicado a la investigación del problema, con el fin de aportar modelos y algoritmos de búsqueda que contribuyan a la correcta y eficiente resolución.

La planificación de enfermeras y en general la planificación de turnos, ha sido tema de investigación desde hace décadas. Comenzó a ser estudiado a partir del año 1980 [20, 32, 27], sin embargo, sólo en la actualidad se ha resuelto con el uso de la programación con restricciones [9, 18, 10]. Esto ha originado que hasta el momento exista una variedad significativa de métodos para resolver el NRP.

En el campo de las heurísticas de ordenamiento y valor para el NRP, en [10, 11, 12] se analizan con la utilización de VNS (Variable Neighbourhood Search). En términos generales, VNS consiste en cambiar sistemáticamente la estructura de entornos, a medida que se realiza la búsqueda. En consecuencia, si alguna heurística de búsqueda falla, para mejorar la solución, el algoritmo escoge de manera dinámica un entorno diferente. En [6] se desarrolló y aplicó un algoritmo para un hospital asiático, con la utilización de programación con restricciones, en particular ocupando la técnica Look-ahead [3], junto con las heurísticas de selección de variable y valor.

Asimismo, en [23] se lleva a cabo una evaluación de dos casos de estudio en los cuales se ha implementado sistemas de turnos para enfermeras. La finalidad es realizar una comparación de diferentes modelos de NRP, mediante la definición de criterios, tales como, la capacidad de representar una amplia variedad de restricciones, la flexibilidad hacia los cambios en el modelamiento, la eficiencia y el poder de aprendizaje del algoritmo. En [17] el autor muestra y compara diferentes métodos para resolver modelos del NRP principalmente basado en la utilización de restricciones globales [1].

4. Programación con restricciones

Todo proceso de resolución de un CSP consta de dos etapas, las cuales deben diferenciarse de forma clara. La primera fase es el modelado del problema en cuestión, como un problema de satisfacción de restricciones, que incluye su representación en términos de variables, dominios y restricciones. Una vez formulado el modelo, la segunda etapa debe realizar un procesamiento, utilizando algoritmos de búsqueda y métodos de consistencia. Ambas técnicas empleadas, ya sea de forma aislada o fusionadas, son incluidas en los *solvers*.

4.1. Formalización de un problema de satisfacción de restricciones

Un problema de satisfacción de restricciones P es definido como una tripleta $P = (X, D, C)$ donde:

- X es un conjunto de variables x_1, x_2, ... , x_n.
- D es un conjunto de dominios d_1, d_2, ..., d_n, donde d_i es el dominio de x_i e $i = 1 \dots n$.
- C es el conjunto de restricciones c_1, c_2, ... , c_m, donde $j = 1 \dots m$, c_j es la relación sobre el conjunto de variables x_1, x_2, ... , x_n y $c(x_i, x_j)$ son restricciones entre x_i y x_j.

4.2. Algoritmos de búsqueda

A continuación se explican los algoritmos utilizados para solucionar un CSP. Para ejemplificar el funcionamiento de cada algoritmo, se considerará el clásico problema de las N-reinas, el cual consiste en posicionar N reinas en un tablero de ajedrez de dimensiones $N * N$, de tal manera que no se amenacen entre ellas. Por reglamento del juego las reinas se mueven en su misma fila, columna y diagonales.

4.2.1. Generate and Test (GT)

El GT es el más básico de los métodos, consiste en instanciar cada una de las variables, comprobando por cada una, si satisface todas las restricciones del problema. El algoritmo va tomando sistemáticamente las posibles soluciones desde el conjunto que corresponde a todas las posibles combinaciones entre los valores de las variables.

GT es un algoritmo poco eficiente, ya que efectúa comprobaciones innecesarias y repetitivas. Una vez finalizada la asignación completa de las variables, comprueba si la asignación satisface todas las restricciones, de manera que sea una solución al problema. Su ineficiencia radica en que esta metodología genera asignaciones completas, que muchas veces transgreden las mismas restricciones.

Por ejemplo, en la figura 1, en el árbol generado por la asignación de las variables en cada una de las posiciones, sólo es posible comprobar si cumple con todas restricciones

al momento de llegar a la parte más baja del árbol. El algoritmo es incapaz de detectar antes si alguna restricción no ha sido satisfecha. Por otro lado, la búsqueda completa, en la mayoría de los casos tiene un alto costo. Por lo que, el uso de GT es recomendable de aplicar sólo a problemas pequeños.

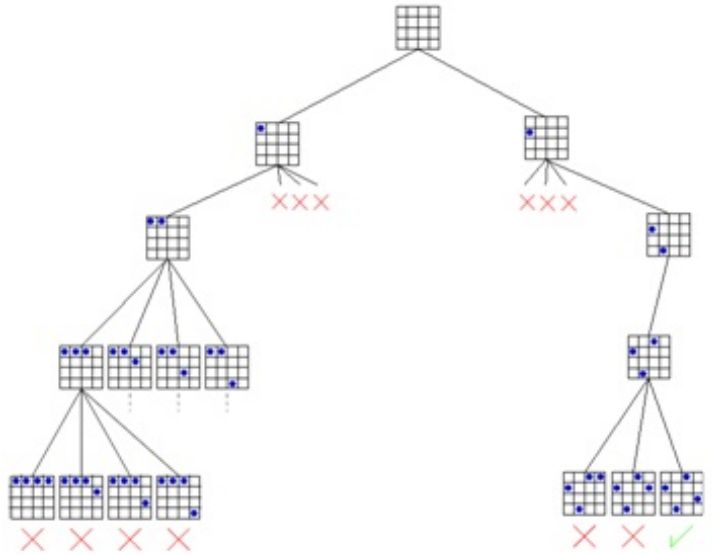

Figura 1: Resolución problema de 4-reinas utilizando GT

4.2.2. Backtracking (BT)

BT es otro enfoque de la exploración y generación de árboles de búsqueda. En este método las soluciones posibles son generadas de forma incremental por la elección repetitiva de un valor para una variable y tan pronto como todas las variables involucradas en una restricción son instanciadas, la restricción es revisada. De esta forma, si una solución parcial viola una restricción, el algoritmo retorna de las más recientes variables instanciadas aquella que todavía tiene alternativas disponibles, eliminando como consecuencia el subespacio en conflicto.

En la figura 2, se muestra el proceso de búsqueda para el problema de 4-reinas, mediante BT. Se aprecia, que se pueden detectar fallas tan sólo con 2 variables instanciadas. No obstante, este método es incapaz de detectar fallas antes de asignar los valores para todas las variables involucradas en una restricción en conflicto. Esta falencia puede ser resuelta, mediante la utilización de técnicas de filtrado.

4.3. Técnicas de consistencia

El término consistencia se refiere a valores individuales o combinación de valores que eventualmente forman parte de la solución, ya que satisfacen las restricciones del CSP.

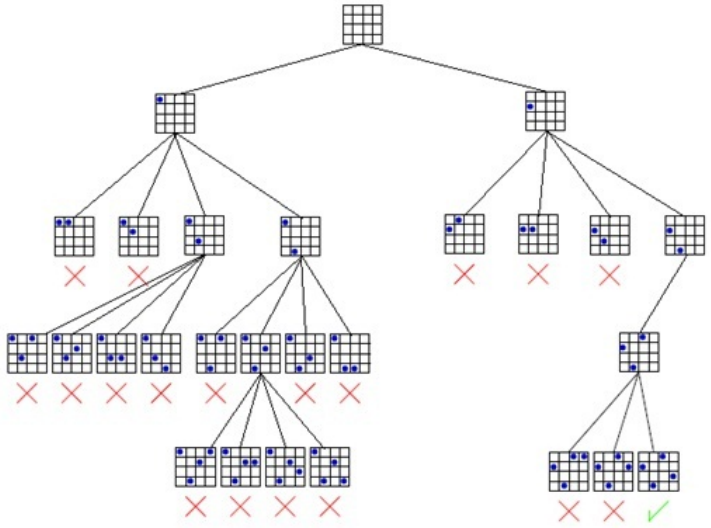

Figura 2: Resolución problema de 4-reinas utilizando BT

Las técnicas de consistencia pretenden reducir el espacio de búsqueda, con un proceso de filtrado que elimina valores que no forman parte de ninguna solución. Dichas técnicas, se clasifican en distintos niveles.

4.3.1. Consistencia de nodo

El funcionamiento de la consistencia de nodo se basa en asegurar que todos los valores en el dominio de una variable satisfagan las restricciones unarias sobre esa variable. Un CSP es nodo consistente si por cada valor del dominio de la variable x, cada restricción unaria sobre x es satisfecha. Si el dominio de la variable x, contiene valores que no son satisfechos, entonces la inconsistencia de nodo puede ser eliminada, ya que no estaría contenida en ninguna solución. Este proceso se realiza eliminando los valores desde el dominio de X.

Sea $P = (X, D, C)$, se dice que P es nodo-consistente si y sólo si:

$$\forall x_i \in X, \forall\, c_j, \exists a \in d_i, \text{ tal que el elemento } a \text{ satisface } c_j$$

Un problema es nodo consistente, si y sólo si todas sus variables son nodo-consistentes.

4.3.2. Consistencia de arco

La consistencia de arco trabaja con restricciones binarias. Un CSP es arco consistente si para cada uno del par de variables restringidas x_i y x_j, para cada valor en a en d_i, existe por lo menos un valor b en d_j, tal que las asignaciones en (x_i, a) y (x_j, b) satisfacen la restricción entre x_i y x_j. Lo anterior conduce a eliminar la consistencia de arco de cualquier valor en el dominio d_i de la variable x_i, que no es arco consistente, ya que no

formaría parte de ninguna solución. El dominio de una variable es arco consistente si todos los valores de dicha variable son arco consistentes.

Sea $P = (X, D, C)$, se dice que P es arco-consistente si y sólo si:

$$\forall\, c(x_i, x_j) \in C,\ \forall a \in d_i,\ \exists b \in d_j,$$
tal que el elemento b es un soporte para el elemento a en $c(x_i, x_j)$

Un problema es arco consistente, si y sólo si todos sus arcos son arco-consistentes.

4.4. Algoritmos de búsqueda utilizando técnicas de consistencia

4.4.1. Forward Checking (FC)

FC es capaz de evitar conflictos futuros mediante la realización de la técnica de arco consistencia en las variables aun no instanciadas. Esto se hace mediante la eliminación temporal de valores a las variables que pueden causar conflicto con la actual variable asignada. Por lo tanto, el algoritmo al detectar que la actual solución es inconsistente, elimina las búsquedas del subárbol usando un simple Backtracking.

La figura 3, muestra como los valores del domino son removidos a partir del segundo nivel del árbol, al posicionar la reina, los futuros valores en conflicto son removidos parcialmente por la reina de posición $(1, 1)$ tanto en la fila 1 como en la diagonal noroeste-sureste. En el árbol de la izquierda la segunda reina es puesta en la posición $(3, 2)$ generando una inconsistencia, esto se debe a que una tercera reina no podría ser ubicada en ningún casillero de la columna 3.

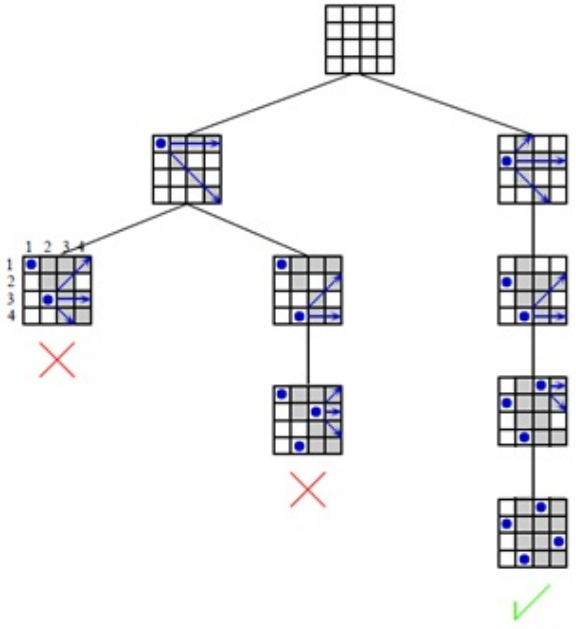

Figura 3: Resolución problema de 4-reinas utilizando FC

4.4.2. Maintaining Arc Consistency (Full Look Ahead)

La técnica de FC, puede ser mejorada a través de un proceso más inteligente, capaz de instanciar los conflictos actuales y los futuros. Este método es llamado Maintaining Arc Consistency (MAC), el cual intenta reducir el espacio de búsqueda de forma más temprana que el FC. No obstante, hace un trabajo mayor en la asignación de cada variable.

Por ejemplo, en la figura 4, cuando la primera reina es colocada en la posición $(1, 1)$, los conflictos entre aquella celda y las futuras posiciones son eliminados, mediante la acotación del dominio de las posiciones posteriores (esto se indica con líneas azules). Luego de eso, desde la posición $(1, 1)$ se comienza a revisar los casos, comenzando en la primera celda disponible en la segunda columna, para el ejemplo esta corresponde a $(3, 2)$. El proceso algorítmico, descubre que aquella posición es inconsistente, ya que desde esa ubicación, no existiría lugar para una tercera reina (flechas anaranjadas), de esta forma, la ubicación $(3, 2)$ es removida. A continuación el algoritmo, sigue con la celda $(4, 2)$, que corresponde a la segunda posición disponible de la segunda columna. Esta posición establece que la única celda disponible para la tercera reina, debe ser la $(2, 3)$, la cual provocaría una inconsistencia, ya que no habría lugar disponible para una cuarta reina. El ejemplo de funcionamiento del algoritmo para la primera posición, se repite de forma equivalente, hasta llegar al resultado de la derecha del árbol de la figura.

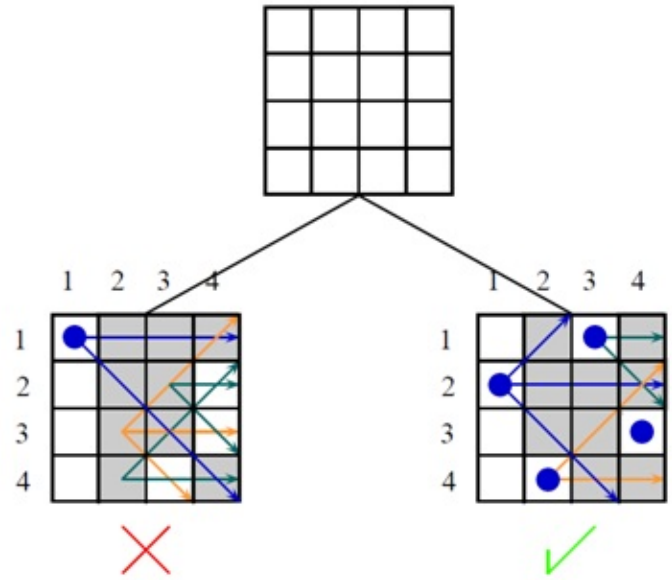

Figura 4: Resolución problema de 4-reinas utilizando MAC

4.5. Heurísticas

El modo en que el algoritmo de búsqueda seleccione el orden en que examinará las variables y el orden en el que éstas serán instanciadas puede provocar una enorme

diferencia en el esfuerzo de búsqueda para lograr solucionar un CSP. A continuación se expondrán las principales heurísticas para la selección de variables y valor.

4.5.1. Ordenamiento de las variables

Establecer cuál de todas las variables será analizada antes que otra, determinará si el método de búsqueda hará más trabajo del necesario. Un factor crítico para los tiempos de búsqueda es el hecho de saber de forma correcta y certera, la variable que será estudiada, debido a que una buena selección ayudará a reducir de forma considerable el espacio de búsqueda. Las heurísticas de ordenamiento tienen la flexibilidad de cambiar el orden en que serán revisadas las variables, dependiendo del estado en que se encuentren los dominios de éstas. Se han propuesto varias heurísticas de ordenamiento de variables, algunas de las más importantes se mencionan a continuación.

- $first - fail$: consiste en seleccionar la variable con el dominio más pequeño. Esta elección se fundamenta por el hecho de que una variable con un dominio reducido, es más probable que falle, por lo tanto, se intenta primero con la variable con la menor cantidad de valores alternativos disponibles. Esto se sustenta en que al elegir una variable más restringida, intuitivamente se piensa que fallará rápidamente. Con esto se restringe el tamaño del árbol. Esta heurística generalmente se adapta mejor a los dominios discretos [25].

- $reduce - first$: en este caso se comienza por seleccionar la variable con el mayor dominio.

- $round - robin$: seleccionar las variables en algún orden, por ejemplo desde la primera variable definida en el modelo hasta la última.

Asimismo, se ha utilizado las técnicas de heurísticas de selección de variables para combinarlos con varios algoritmos de búsqueda, creando diversas extensiones. El estudio completo y detallado se expone en [13].

4.5.2. Selección de los valores

Una vez seleccionada la variable, el algoritmo tiene que escoger que valor se elegirá desde el dominio de la variable. Fundamentalmente lo que se busca es instanciar un valor de la variable, de modo que conduzca lo más rápido a la solución. El orden en que se instancian, puede tener un impacto considerable en el tiempo requerido para encontrar la primera solución.

Una selección intuitiva corresponde a elegir la variable a partir de su valor numérico, esto es, seleccionar el máximo valor, mínimo valor o el valor medio. Qué valor será escogido dependerá del escenario del problema. Por ejemplo, en el problema de las N-reinas (presentado anteriormente), es conveniente elegir el valor medio ya que reduce más rápidamente los dominios de las siguientes variables. En otras palabras, la propagación de restricciones es más efectiva [1].

Las heurísticas en general seleccionan valores que disminuyen en menor cantidad el número de valores que puedan ser utilizados por las variables restantes. Algunos de los métodos que hacen lo anterior son:

- $min-conflicts$ [29]: selecciona el valor de la variable que provocará menos conflictos con las variables que serán instanciadas. Se elegirá el valor que genere menor incompatibilidad con las futuras variables.
- $max - domain - size$ [14]: esta heurística selecciona el valor de la variable actual que deja el máximo dominio en las variables futuras.

5. *Solvers*

Fueron creados para desarrollar y resolver CSP, el primero surgió alrededor del año 1960, siendo utilizados hasta la actualidad para desarrollar problemas que involucran variables, dominios y restricciones. Para soportar restricciones y diversos paradigmas, han sido creados variados tipos de *solvers* de diversa naturaleza. Si bien existen distintas clasificaciones, se dará énfasis en explicar y ejemplificar tres categorías: los que utilicen la programación lógica; aquellos que utilizan librerías; y por último los lenguajes de modelado.

5.1. Programación lógica con restricciones

La programación lógica con restricciones (CLP, Constraint Logic Programming) es el paradigma que une la programación lógica y la satisfacción de restricciones. El lenguaje utilizado se caracteriza por tener una estructura algebraica. Las funciones especiales y predicados simbólicos se interpretan sobre un dominio fijo, toda relación que se establezca con el dominio es denominada restricción.

Esta idea fue impulsada por Colmerauer en el desarrollo de Prolog II [7], luego fue generalizado en el esquema CLP realizado por Jaffar and Lassez [16]. Algunos ejemplos de *solvers* que soportan CLP se nombran a continuación.

5.1.1. ECL^{i}PS^e

ECL^{i}PS^e [19] es el más reciente sistema CLP, proporciona una amplia gama de características para solucionar problemas con restricciones tales como manejo de listas, arreglos y registros, además de un conjunto de instrucciones de control como las condicionales y las repetitivas. ECL^{i}PS^e también provee librerías con la finalidad de tratar cierto tipo de problemas como los que involucran dominios continuos o programación matemática, que además se pueden combinar para resolver problemas en forma híbrida.

5.1.2. GNU Prolog

Es otro sistema que pertenece al grupo de CLP [8], ha sido diseñado para tratar problemas con dominios finitos, sin embargo, también se puede utilizar para trabajar con números reales. Tiene entre sus herramientas una larga lista de predicados predefinidos de Prolog y restricciones para el manejo de listas y estructuras condicionales. Se ha incluido soporte para problemas de optimización y heurísticas de ordenamiento, además de una interfaz para llamar a rutinas externas escritas en el lenguaje C.

5.2. Librerías

Las librerías proporcionan un lenguaje necesario para enunciar problemas bajo restricciones al haber limitaciones con algún lenguaje de programación. Por lo general son implementadas por medio de clases y métodos específicos. Por ejemplo, una determinada clase puede ser usada para definir el estado de las variables y los métodos, junto con las

relaciones entre ellos. Esta es una forma de implementar un sistema de restricciones sin la necesidad de desarrollar un nuevo lenguaje de programación. No obstante, el usuario está obligado a conocer el lenguaje nativo para la determinada librería que desee utilizar. Algunos *solvers* que pertenecen a esta categoría se exponen a continuación.

5.2.1. Generic Constraint Development Enviroment (Gecode)

Gecode [30] es una librería escrita sobre C++. Fue diseñada para soportar variables de dominio finito. El conjunto de restricciones que abarca, permite trabajar sobre tipos de datos enteros, booleanos y conjuntos de variables, además soporta la definición de heurísticas de ordenamiento de variable y selección de valor. Entre algunos de los lenguajes en los cuales se han desarrollado interfaces para trabajar con librerías Gecode se encuentran Prolog (YAP Prolog), Java (Gecode/J), Ruby (Gecode/R), Lisp (GeLisp) y Phyton.

5.2.2. Koalog

Koalog [31] es una librería perteneciente al lenguaje Java que permite trabajar con problemas relacionados con satisfacción y optimización de restricciones. Admite definir conjuntos de restricciones y dominios, ambos finitos. Entre sus cualidades, soporta especificación de heurísticas de selección de variables y valor. Además posee mecanismos de búsqueda que pueden ser declarados mediante objetos específicos.

5.2.3. Choco

Es un *solver* de programación con restricciones desarrollado como una librería de Java. Provee un variado conjunto de restricciones para ser aplicadas sobre números enteros, reales, y conjunto de variables. Soporta problemas de optimización y el proceso de búsqueda puede ser realizado por estrategia de selección de variables y valor, predefinida por el *solver* o definidas por el usuario.

5.3. Lenguajes de modelado

El propósito de los sistemas con lenguaje de modelado apunta a simplificar la definición de problemas basado en restricciones. Intentan alejar al usuario de la complicada codificación aún presente en típicas librerías o lenguajes de programación. El centro del lenguaje en general es más comprensible porque proporciona una semántica y sintaxis simplificada. En estos lenguajes, dependiendo del enfoque utilizado existen casos donde se permite la especificación de procedimientos de búsqueda.

5.3.1. OPL

OPL [15] es uno de los principales lenguaje de modelado. Su sintaxis y semántica ha sido usada como base de modernos lenguajes de modelado. El lenguaje OPL está compuesto por varios constructores de alto nivel, por ejemplo, estructuras de datos,

tales como, arreglos, variables de dominio finito, ciclos y estados condicionales, y un conjunto de órdenes internas para la asignación de recursos. También permite heurísticas de ordenamiento de variables y selección de valores. Una característica interesante de OPL y tal vez su principal novedad, es que las estrategias de búsqueda pueden ser especificadas usando el mismo estilo utilizado para plantear el problema.

5.3.2. Zinc

Zinc [26] es uno de los más recientes lenguajes de modelado. Posee una sintaxis que soporta los predicados definidos por el usuario y las funciones. Está provisto de estructuras de datos, conjuntos, abstracciones de control y dominios finitos y continuos. La plataforma se apoya en una arquitectura *solver-independiente*, donde los modelos de Zinc pueden ser asignados a tres modelos de ECL^{i}PS^e: modelo de programación con restricciones, modelo de búsqueda local y modelo de programación matemática. Para facilitar la traducción a otros modelos se incluye un modelo intermedio llamado FlatZinc.

5.3.3. MiniZinc

MiniZinc [21] es una versión más pequeña de Zinc, donde el usuario define tipos y funciones, pero algunas restricciones han sido excluidas. Minizinc también se basa en una arquitectura superior independiente del *solver* de solución, permitiendo asignaciones desde MiniZinc a ECL^{i}PS^e y Gecode. El proceso de asignaciones se realiza por medio de ciertas condiciones de reescritura, basadas en un sistema de transformación llamado Cadmium [28], el cual permite especificar la traducción desde la fuente al modelo destino. Al igual que en Zinc, para facilitar la traducción se utiliza el modelo intermedio FlatZinc.

6. ECL^{i}PS^e

A continuación se expondrán características fundamentales de ECL^{i}PS^e, el cual en una sección posterior será el sistema utilizado para la resolución del NRP.

ECL^{i}PS^e es un sistema de software de código abierto, basado en CLP. Este último, combina dos paradigmas de programación: la programación lógica y la programación con restricciones. Contiene una colección de librerías para la resolución de restricciones y un modelamiento de alto nivel, a través de un lenguaje basado en Prolog [2].

El sitio web principal de ECL^{i}PS^e ($www.eclipse - clp.org$) proporciona una amplia cantidad de recursos para ayudar al usuario en el desarrollo de sus aplicaciones, incluyendo manuales, referencias bibliográficas y un índice de librerías continuamente actualizadas. Además contiene una variada colección de problemas resueltos, tales como, el N-reinas, sudoku, cuadrados mágicos y problemas de planificación.

6.1. Librerías

Se mostrarán características de las principales librerías utilizadas en ECL^{i}PS^e.

6.1.1. Implementación de dominios y restricciones

ic

El estándar de resolución de restricciones ofrecidas por la mayoría de los sistemas de programación con restricciones, es el basado en dominios finitos. ECL^{i}PS^e da soporte a dominios finitos a través de la librería ic y fd. Esta librería implementa dominios finitos de números enteros, junto con un conjunto básico de restricciones [22]. Además, ic permite dominios continuos los cuales deben ser representados dentro de intervalos numéricos.

suspend

Se utiliza para incluir restricciones matemáticas como mayor o igual (>=), mayor(>), igual (=:=), distinto (= \ =), menor o igual (<=) y menor (<).

ic_global

Esta librería soporta una variedad de restricciones, cada una de las cuales son tomadas como un argumento de una lista de variables de dominios finitos, son denominadas restricciones globales [4]. Algunos ejemplos disponibles en la librería ic_global son:

- $alldifferent(Lista)$: obliga a que todos los elementos de la lista sean diferentes entre sí.
- $maxlist(Lista, Max)$: Max es el máximo de los valores la $Lista$.
- $occurrences(Valor, Lista, N)$: la variable $Valor$ aparecerá exactamente N veces en la colección $Lista$.

ic_cumulative, ic_edge_finder

Son utilizadas para restricciones de planificación. Existen diversas librerías de ECL^{i}PS^e que implementan restricciones globales para aplicaciones de planificación. La restricción toma una lista de tareas (hora de comienzo, duración y recursos necesarios) y un nivel máximo de recursos. Las tres librerías de ECL^{i}PS^e que implementan restricciones de planificación son *ic_cumulative*, *ic_edge_finder* y *ic_edge_finder*3, las cuales tienen las misma estructura de declaración de sus restricciones, pero se diferencian en el manejo de la complejidad de los tiempos y el poder de propagación.

ic_sets

Implementa restricciones sobre un conjunto finito de enteros. Las restricciones son relaciones comunes entre los conjuntos. Por ejemplo, inclusión, intersección, unión y disyunción. Además existen restricciones, como la cardinalidad, aplicada entre conjuntos y enteros, .

eplex

Esta librería soporta una estrecha integración entre la programación lineal y la programación entera.

ic_simbolic

Permite el manejo de restricciones sobre variables que pertenecen a un dominio ordenado. Por ejemplo, nombre de productos o nombre de los días de una semana.

6.1.2. Restricciones definidas por el usuario: *propia*

Esta librería maneja reglas de propagación sobre un predicado. El predicado $infers/2$, toma como primer argumento cualquier predicado definido por el usuario y como segundo argumento la forma de propagación que debe ser aplicado sobre ese predicado. La forma de propagación incluye dominios finitos e intervalos.

6.2. Ciclos

ECL^{i}PS^e admite el uso de ciclos iterativos sobre listas, procesamiento de listas y elementos atómicos, con los siguientes predicados:

- $fromto(First, In, Out, Last)$: itera comenzando con $In = First$, hasta $Out = Last$.
- $foreach(X, List)$: itera con X abarcando todos los elementos de la lista $List$.

- $foreacharg(X, StructOrArray)$: itera con X abarcando todos los elementos de un arreglo.
- $for(I, MinExpr, MaxExpr)$: hace una iteración de I, que comienza desde $MinExpr$ hasta $MaxExpr$. I es una variable local del ciclo, $MinExpr$ y $MaxExpr$ pueden ser expresiones aritméticas. Sólo pueden ser usados para el control de la iteración, por lo que $MaxExpr$ y $MinExp$ no pueden ser instanciadas.

Muchas veces en estos ciclos, es necesario incluir el predicado $param(Var1, Var2, ...)$ que se utiliza para declarar que las variables sean globales dentro de un contexto, por ejemplo, para que una variable sea compartida en todas la iteraciones de un ciclo.

6.3. Ejemplo N-reinas

En la figura 5, se muestra la resolución del problema de N-reinas en ECL^{i}PS^e. Se comienza declarando las librerias *ic* y *suspend*, junto con definición del predicado `queens` (línea 4) que genera la solución *QueenStruct* de dimensión *Number*. Dentro de `queens` el predicado `constraint` (línea 8) se encarga de validar las restricciones aritméticas para cada posición definida por j e i de la lista *QueenStruct*. Por su parte el predicado de búsqueda `search` (línea 17) selecciona un valor dentro de la lista *QueenStruct* de dimensión *Number*.

```
1  :-lib(ic).
2  :-lib(suspend).
3
4  queens(QueenStruct,Number):-
5   dim(QueenStruct,[Number]),
6   constraint(QueenStruct,Number),
7   search(QueenStruct).
8  constraint(QueenStruct,Number):-
9   (for(I,1,Number),param(QueenStruct,Number)do
10   QueenStruct[I]::1..Number,
11   (for(J,1,I-1),param(I,QueenStruct)do
12    QueenStruct[I] $\= QueenStruct[J],
13    QueenStruct[I]-QueenStruct[J] $\= I-J,
14    QueenStruct[I]-QueenStruct[J] $\= J-I
15   )
16  ).
17 search(QueenStruct):-
18  dim(QueenStruct,[N]),
19  (foreacharg(Col,QueenStruct),param(N) do
20   select_val(1,N,Col)
21  ).
```

Figura 5: Resolución N-reinas en ECL^{i}PS^e

7. Estudio del NRP

7.1. Descripción del problema

El NRP consiste en la asignación de turnos a un número finito de enfermeras, de tal manera que la planificación satisfaga un conjunto de restricciones en un periodo de tiempo determinado. Las restricciones se originarán según requerimientos de distinta naturaleza como tipos de turnos, preferencias de las enfermeras, cantidad de días de la planificación y políticas del hospital.

La asignación de turnos en los sistemas de salud ha sido un desafío pendiente para los sistemas de asignación automática. Mientras que en otras áreas sería posible aceptar la creación de lista de turnos de baja calidad de atención, en los sistemas de listado de enfermeras no sería factible, ya que es un componente crítico de la organización. Los hospitales deben evitar que las enfermeras estén con altos niveles de stress, cansancio o sobrecargadas de trabajo. Esto influiría de forma directa en la calidad del cuidado de los pacientes.

La creación de un listado es sin duda una tarea compleja. Muchas restricciones deben tomarse en consideración y deben quedar balanceadas con cuidado de acuerdo a su importancia. En muchos lugares estos listados son hechos a mano sin un proceso automático, por lo que consume al mes una gran cantidad de tiempo. Para la creación de un listado de turnos se debe tomar en cuenta los diferentes tipos de restricciones:

- Regulaciones legales: una correcta asignación debe fundamentarse en las leyes que regulan cada caso de estudio. Factores a considerar son los relacionados con el máximo y mínimo de tiempo trabajado por día o por semana por cada persona, cantidad mínima de horas libres entre turnos o licencias laborales.

- Reglas organizacionales: son aquellas que se aplican en específico a un centro de salud en particular, a una área o incluso sólo a una sala del hospital. Están dadas por las respectivas administraciones. Son las relacionadas con el número y tipos de turnos que se manejan y el mínimo de personas ubicadas en cada área, piso o sala. Por ejemplo, en algunos hospitales se manejan tres tipos de turnos: mañana (7:00 hrs-15:00 hrs), tarde (15:00 hrs-23:00 hrs) y noche (23:00 hrs-7:00 hrs). Sin embargo, en otros centros de salud su administración se basa en sólo dos turnos: día y noche. Y en relación a la cantidad de personal para cada turno, también existen variaciones significativas.

- Datos del personal: acá se define un marco individual para cada persona. Se trata de obligaciones contractuales del tiempo de trabajo, vacaciones pendientes y horas extraordinarias acumuladas. Si, por ejemplo, una enfermera tiene 20 horas extraordinarias, debería ser asignada dos turnos menos que el promedio.

- Requerimientos del personal: son requerimientos dados por el personal. Estos esencialmente corresponden a deseos de tener unos días de descanso, por ejemplo, en fines de semana, feriados o para un periodo de vacaciones.

Comúnmente las restricciones se categorizan dentro de dos grupos: restricciones duras (hard constraints) las cuales deben ser satisfechas en su totalidad; y restricciones blandas (soft constraints) que corresponden a aquellas donde el cumplimiento de ciertas restricciones no es necesario que sea total, existiendo un cierto grado de flexibilidad.

8. Resolución NRP preliminar en ECL^{i}PS^e

8.1. Caso de estudio

Para la formalización del NRP se visitó la Clínica Valparaíso a fin de tener una visión real de la planificación de turnos que ocupa la enfermera jefa para su personal a cargo. Antes de exponer el sistema de turno se presentará información básica de la clínica, de tal manera de contextualizar el caso de estudio.

Clínica Valparaíso S.A es una institución de salud privada perteneciente Red de Clínicas Regionales de la "Mutual de Seguridad de la Cámara Chilena de la Construcción" y de la Asociación Chilena de Seguridad ACHS. Comenzó sus actividades el 1° de diciembre de 2004. A continuación se describen aspectos relacionados con la distribución de las áreas clínicas, médicos, enfermeras, sistemas de turnos y la terminología ocupada.

Áreas clínicas

Las áreas centrales de la institución corresponden a las siguientes:

- Urgencia
- Centro médico
- Pabellón
- Unidad de cuidados intensivos (UCI)
- Hospitalización (médico-quirúrgico): respecto a la información relacionada con el número de enfermeras y paramédicos, para la sección de hospitalización y centro médico, se cuenta con:
  - Pabellón e infección hospitalaria: 1 enfermera jefe por cada área
  - Urgencia: 4 enfermeras y 8 paramédicos
  - UCI: 4 enfermeras y 8 paramédicos
  - Recuperación: 1 enfermera y 4 paramédicos
- Centro médico: sólo lo integran paramédicos.

Médicos y atenciones

En relación al número de atenciones médicas y cantidad de médicos para las distintas áreas, se cuenta con lo siguiente:

- Promedio de número de egresos hospitalarios mensuales: en hospitalización 200 personas y en consultas de urgencia, 2000 personas
- Médicos de urgencia contratados: en área pediatría 5 y en área adultos 5 personas

- Médicos visitantes: 50

Observaciones sobre cambios de turnos

A continuación se muestran las observaciones que fueron rescatadas de la visita al centro de salud y que eventualmente podrían repercutir en el modelado del problema.

- Vacaciones y cambios de turno son coordinados entre enfermera jefe y el reemplazante, en ningún caso se cambiará la estructuras de los turnos
- Nunca un paramédico podrá reemplazar una enfermera
- En la UCI no hay cambios, ni reemplazos entre enfermeras de otras áreas, ya que corresponden a especialidades médicas

Terminología

Con respecto a la terminología utilizada por la clínica, se rescataron conceptos que serán empleados para la investigación. Clínica Valparaíso utiliza el denominado sistema del cuarto turno, que es ocupado por la mayoría de los centros de salud nacionales, el cual se organiza de la siguiente manera:

- Larga: corresponde a un turno de día que va desde las 8:00 AM hasta las 08:00 PM.
- Noche: corresponde a un turno de noche que va desde las 8:00 PM hasta las 08:00 AM.

La asignación de una enfermera debe mantener el siguiente orden, una vez concluido debe repetir:

- Día 1: Larga
- Día 2: Noche
- Día 3: Libre
- Día 4: Libre

En la tabla 1, se puede observar el correcto funcionar de la asignación. Donde una enfermera ingresa a trabajar el lunes a las 8 AM, el martes continúa de noche a las 8 PM y el miércoles y jueves serán libres. Esto provoca un ciclo para cada enfermera, es decir, el día viernes debe trabajar durante el día.

DIA 1		DIA 2		DIA 3		DIA 4	
Lunes		Martes		Miércoles		Jueves	
8AM - 8PM	8PM - 8AM	8AM - 8PM	8PM - 8AM	8AM - 8PM	8PM - 8AM	8AM - 8PM	8PM - 8AM
Larga			Noche				

Tabla 1: Funcionamiento del cuarto turno

8.2. Definición del modelo para el NRP preliminar

Para una enfermera i en un día j, la variable $V_{i,j}$ puede tener uno de los siguientes valores:

- $V_{i,j} = 0$**:** indica que la enfermera i esta libre el día j
- $V_{i,j} = 1$**:** indica que la enfermera i esta asignada para el turno larga en el día j
- $V_{i,j} = 2$**:** indica que la enfermera i esta asignada para el turno noche en el día j

La tabla 2, muestra una asignación para 8 enfermeras durante 28 días. Las filas contienen el turno de cierta enfermera y las columnas contienen los turnos realizados en un cierto día. Cada celda determina el día y el turno en que debe trabajar cada enfermera, por ejemplo, la enfermera 2 el día 3 debe trabajar en turno larga.

	DIAS																		
ENFERMERAS	1	2	3	4	5	6	7	8	9	...	20	21	22	23	24	25	26	27	28
1	0	0	1	2	0	0	1	2	0	...	2	0	0	1	2	0	0	1	2
2	0	0	1	2	0	0	1	2	0	...	2	0	0	1	2	0	0	1	2
3	1	2	0	0	1	2	0	0	1	...	0	1	2	0	0	1	2	0	0
4	2	0	0	1	2	0	0	1	2	...	1	2	0	0	1	2	0	0	1
5	0	1	2	0	0	1	2	0	0	...	0	0	1	2	0	0	1	2	0
6	2	0	0	1	2	0	0	1	2	...	1	2	0	0	1	2	0	0	1
7	0	1	2	0	0	1	2	0	0	...	0	0	1	2	0	0	1	2	0
8	1	2	0	0	1	2	0	0	1	...	0	1	2	0	0	1	2	0	0

Tabla 2: Asignación para 8 enfermeras durante 28 días

Referente al tamaño de la asignación, la constante que determina el número total de enfermeras se denotará como $totalEnf$ y con respecto a los días, con el fin de evitar la redundancia que ocurre durante todo el año, la cantidad que se acomoda de mejor manera es 28 días. Este número corresponde al mínimo común múltiplo entre la cantidad de días de la semana (7) y el ciclo de 4 días que debe cumplir cada enfermera. Por ejemplo, si la Enfermera 1, entra al turno larga el día 1 (lunes), terminará el ciclo con su segundo día libre el día 28 (domingo).

Con esta notación se pueden escribir todas las variables del modelo de la siguiente manera:

$$[V_{1,1},\, V_{1,2},\, \ldots,\, V_{i,j}] \in [0,2],\, tal\, que\, i \in [1, totalEnf],\, j \in [1,28]$$

Definidas las variables y restricciones la formalización del modelo matemático se expresa a continuación.

- **Variables**

  $< V_{1,1}, V_{1,2}, \ldots, V_{i,j} > \in [0,2], \, tal\, que\, i \in [1, totalEnf], \, j \in [1,28]$

- **Constantes**

  $TotalEnf$

- **Restricciones**

  - $Para\, i \in [1, totalEnf] \, \wedge \, j \in [1, 28-2]$

    1. $[(V_{i,j} = 0 \, \wedge \, V_{i,j+1} = 0) \, \Rightarrow \, (V_{i,j+2} = 1)]$
       Esta restricción establece que después de dos días libres, siga un turno larga.
    2. $[(V_{i,j} = 2) \, \Rightarrow \, (V_{i,j+1} = 0 \, \wedge \, V_{i,j+2} = 0)]$
       Esta restricción establece que después de un turno de noche sigan dos días libres.

  - $Para\, i \in [1, totalEnf] \, \wedge \, j \in [1, 28-1]$

    3. $[(V_{i,j} = 1 \, \wedge \, V_{i,j+1} = 2)]$
    4. $[(V_{i,j} = 2 \, \wedge \, V_{i,j+1} = 0)]$

       Estas restricciones establecen que después de un turno larga, continúe uno de noche y que después de un turno noche siga uno libre.

  - $Para\, k \in [1, TotalEnf/8] \, \wedge \, j \in [1,28] \, \wedge \, \, i \in [((k-1)*8)+1, ((k-1)*8)+8]$

    5. $[occurrences(0, V_{i,j}, 4)]$
    6. $[occurrences(1, V_{i,j}, 2)]$
    7. $[occurrences(2, V_{i,j}, 2)]$

       Estas restricciones establecen la cantidad de enfermeras diarias de turnos libres (0), larga (1) y noche (2) para los 28 días.

En cuanto a los dominios involucrados en las restricciones 5, 6 y 7 del presente modelo, por un lado j recorre cada uno de los días del periodo de trabajo; k se encarga de contar los grupos posibles de 8 enfermeras, esto ayuda de gran forma a contabilizar las ocurrencias de estas restricciones; finalmente, i se encarga de seleccionar cada una de las enfermeras del respectivo grupo de 8 enfermeras k.

Una vez obtenido el modelo del NRP, la siguiente etapa consiste en implementarlo en el *solver* ECL^{i}PS^e. A continuación se explicará la implementación del modelo matemático en dicho *solver*.

8.3. Codificación del NRP preliminar en ECL^{i}PS^e

De igual manera como fue explicado el código del problema N-reinas, se explicará el código correspondiente al modelo implementado en ECL^{i}PS^e para el NRP.

La figura 6, comienza declarando las librerías *ic* y *ic_global*, esta última requerida para el manejo de restricciones globales, tal como *occurrences*. También se visualiza el predicado principal `nrp` (línea 3), el cual contiene los argumentos de la matriz de resultados, el total de enfermeras y el número total de días. Dentro del mismo predicado se establecen los dominios, los cuales corresponden a la disposición de turnos, es decir, dominio entre 0 y 2 (0: turno Libre, 1: turno Larga y 2: turno Noche).

```
1 :-lib(ic).
2 :-lib(ic_global).
3 nrp(Matriz, TotalEnf, TotalDias) :-
4  dim(Matriz, [TotalEnf, TotalDias]),
5  Matriz[1..TotalEnf,1..TotalDias] :: 0..2,
```

Figura 6: Sección 1 codificación NRP en ECL^{i}PS^e

Continuando con el predicado `nrp`, la figura 7 representa las restricciones para generar la secuencia obligatoria que la clínica mantiene a lo largo de todo el año.

Ambos conjuntos de ciclos *for*, recorren hasta el total de enfermeras y pasan como parámetros al ciclo siguiente la cantidad de días para realizar el siguiente *for*, se realiza un cambio en el índice j, puesto que ambas restricciones quedarían con un total de 30 y 29 días respectivamente.

El siguiente paso es verificar las ubicaciones para situar según sea la restricción el número adecuado para satisfacer el modelo (número correspondiente al turno). El primer conjunto del ciclo *for* contiene las restricciones 1 y 2 del modelo matemático, donde se verifica que al haber dos 0, debe continuar un 1 y si existe un 2 deben continuar dos 0. Caso similar en el segundo conjunto donde se incluyen las restricciones 3 y 4, sólo que, si existe un 1 en una determinada posición debe haber un 2 en la siguiente o si es un 2 que le siga un 0,

La figura 8 muestra el código correspondiente a las restricciones 5, 6 y 7 del modelo, las cuales establecen el total de turnos que deben existir por día. Un primer ciclo *for* recorre la matriz en bloques de 8 enfermeras y el siguiente *for* hace un recorrido por

```
(for(I,1,TotalEnf),param(TotalDias,Matriz) do
  (for(J,1,TotalDias-2), param(I,Matriz) do
   (Matriz[I,J]#=0 and  Matriz[I,J+1]#=0) => Matriz[I,J+2]#=1  ,
  Matriz[I,J]#=2 => (Matriz[I,J+1]#=0 and Matriz[I,J+2]#=0)
  )
 ),
 (for(I,1,TotalEnf),param(TotalDias,Matriz) do
  (for(J,1,TotalDias-1), param(I,Matriz) do
   Matriz[I,J]#=1 => Matriz[I,J+1]#=2,
   Matriz[I,J]#=2 => Matriz[I,J+1]#=0
  )
 ),
```

Figura 7: Sección 2 codificación NRP en ECL^{i}PS^e

cada día. La variable *Columna* almacenará la columna con el total de enfermeras del día dentro de una lista y luego se encargará que en cada columna existan las ocurrencias 0, 1 y 2 un total de 4, 2 y 2 respectivamente, con la restricción global *occurrences*.

```
 (for(K,1,TotalEnf-7,8),param(Matriz,TotalDias) do
  (for(J,1,TotalDias), param(Matriz,K) do
   Columna is Matriz[K..K+7,J],
   occurrences(0,Columna,4),
   occurrences(1,Columna,2),
   occurrences(2,Columna,2)
  )
 ),
 labeling(Matriz),
 print_board(Matriz,TotalEnf,TotalDias).
```

Figura 8: Sección 3 codificación NRP en ECL^{i}PS^e

En la figura 9, se muestra el predicado que se encargará de generar la matriz con las restricciones ya aplicadas e imprimir dicha matriz con la solución por pantalla.

8.4. Compilación y ejecución del NRP en ECL^{i}PS^e

Se utiliza TkECLiPSe para la compilación y posterior ejecución del código fuente explicado en la sección anterior. TkECLiPSe proporciona una interfaz gráfica de alto nivel, con múltiples ventanas, botones y herramientas que ayudan al usuario al desarrollo de código ECL^{i}PS^e. En la figura 10, se muestra la vista inicial del programa TkECLiPSe en su versión 6.0.

Para comenzar se realiza la compilación del código fuente (ver Anexo A). En la figura 11, se muestran los resultados de la compilación de la codificación del NRP. Con la compilación correcta, se ingresa el predicado principal en el área `Query Entry` que en

```
print_board(Matriz,TotalEnf,TotalDias) :-
 dim(Matriz, [TotalEnf,TotalDias]),
(for(I,1,TotalEnf), param(Matriz,TotalDias) do
 (for(J,1,TotalDias), param(Matriz,I) do
   X is Matriz[I,J],
   printf(" %d", [X])
  ), nl
 ), nl.
```

Figura 9: Sección 4 codificación NRP en ECL^{i}PS^e

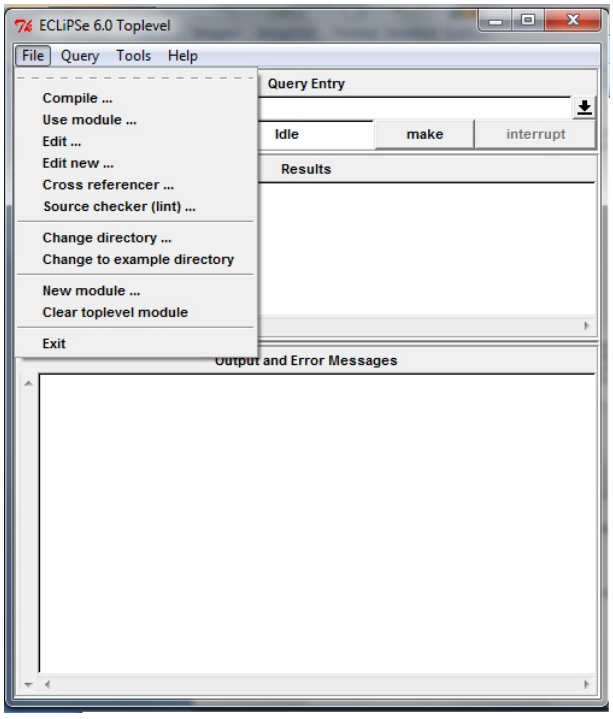

Figura 10: TkECLiPSe 6.0

este caso corresponde a `nrp`($Matriz, 16, 28$). En la figura 12, se muestra el resultado de la matriz de asignación para 16 enfermeras durante 28 días.

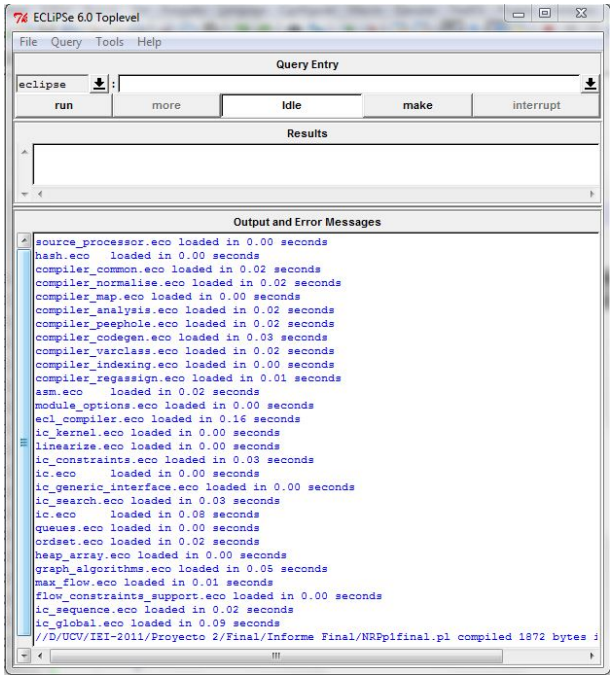

Figura 11: Resultado compilación del NRP en TkECLiPSe

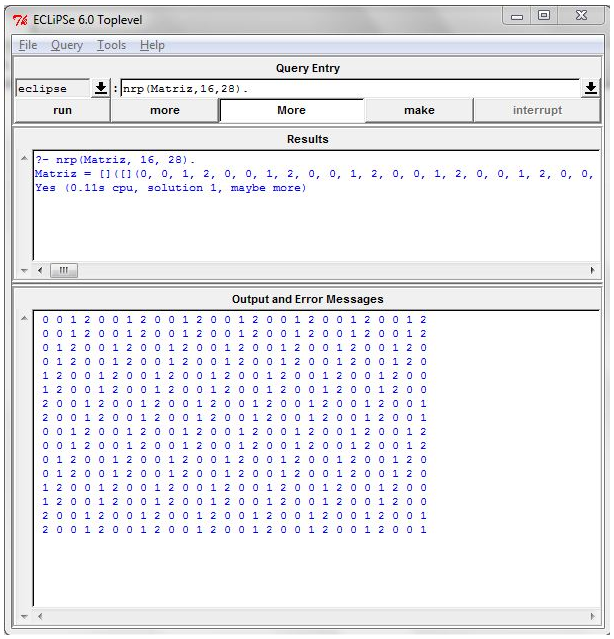

Figura 12: Resultado ejecución del NRP en TkECLiPSe

9. Resolución NRP avanzado en ECL^{i}PS^e

9.1. Descripción del modelo

El modelo avanzado se rige bajo las mismas normas y restricciones del modelo preliminar. Tanto la secuencia de enfermeras, como el significado de los turnos larga, noche y libre, 1, 2 y 0 respectivamente, seguirá siendo el mismo para este nuevo modelo. La diferencia radica en la integración de un nuevo rol, denominado enfermera jefe o supervisora.

El nuevo rol ha sido integrado al modelo con la finalidad de obtener un modelo aún más próximo a la realidad. La importancia que posee dentro del sistema de trabajo de la clínica es imprescindible, ya que es la encargada del trabajo administrativo, de solicitar materiales de enfermería, de organizar los trabajos específicos (curaciones, aseo de pacientes, inyecciones, entre otras) y distribuir el trabajo de las enfermeras a cargo.

El rol de enfermera jefe está sujeto a restricciones propias y su única relación con el sistema del cuarto turno corresponde a la distribución de los turnos por cada día. Además la cantidad de enfermeras a cargo puede ser variable, pero para este modelo, dicha cantidad es dos enfermeras jefes, una para turno larga y otra para turno noche. A continuación se listan características nuevas y otras que modifican el modelo expuesto en la sección anterior.

9.2. Definición del modelo para el NRP avanzado

Se modificaron las dimensiones de la matriz que contendrá la solución. En el modelo preliminar las filas de la matriz correspondían al total de enfermeras tradicionales, sin embargo, en el modelo avanzado a $TotalEnf$ se le suma las enfermeras jefes.

La cantidad de enfermeras jefes corresponde a dos por cada ocho enfermeras tradicionales, ya que los únicos turnos laborales posibles dentro del sistema son larga y noche, los cuales deben ser cubierto a diario (deben trabajar dos enfermeras jefe por cada ocho enfermeras tradicionales). Un ejemplo con la distribución de la nueva matriz para 16 enfermeras y 4 jefes se visualiza en la figura 13.

En la imagen se pueden apreciar las diferencias con la matriz del modelo preliminar, se incluye una nueva variable y nuevas constantes, además de las nuevas restricciones que relacionan ambos tipos de enfermeras, por su parte las nuevas enfermeras poseen restricciones propias que son diferentes a las del resto del personal, mientras que para las enfermeras tradicionales se ha incluido un nuevo criterio. Todos los nuevos cambios se describen a continuación.

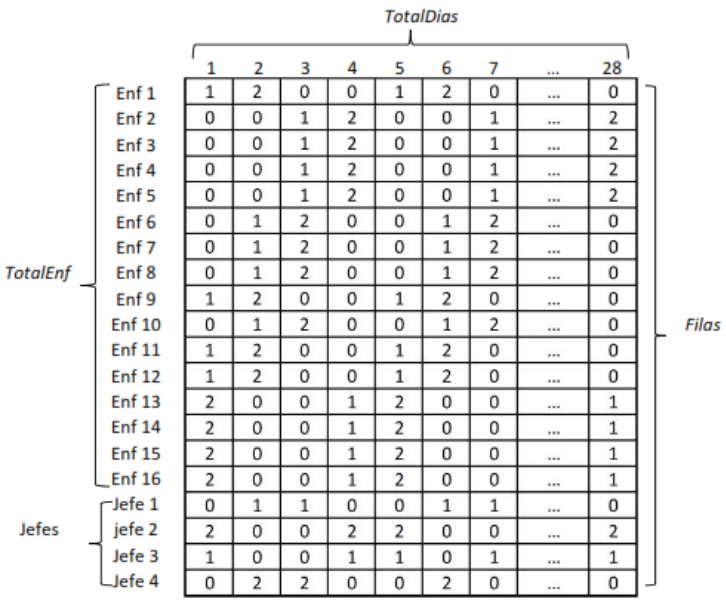

Figura 13: Matriz para modelo avanzado

Constantes

A partir de la figura 13, se pueden apreciar las nuevas constantes las que se suman a $TotalEnf$ del modelo preliminar. Sus significados y notación son los siguientes:

- $Filas$**:** sirve para enumerar el máximo de filas que tendrá la matriz de enfermeras, considerando las enfermeras de turno utilizadas en el modelo inicial y las enfermeras jefes, su valor corresponde a $TotalEnf + TotalEnf/4$.

- $Jefes$: sirve para indicar la posición inicial de las enfermeras jefes en la matriz, su valor inicial corresponde siempre a $TotalEnf + 1$.
- $NJefes$: no representa una posición en la matriz, sino que representa la cantidad de bloques de enfermeras (cada bloque compuesto por ocho enfermeras tradicionales o dos enfermeras jefes) que está definido por $TotalEnf/8$. Sirve para recorrer de mejor manera la matriz e identificar claramente los bloques en las restricciones.

Variables

Al igual que el modelo anterior, para una enfermera i en un día j, una variable $V_{i,j}$ será instanciada con los siguientes valores:

- $V_{i,j} = 0$: indica que la enfermera i esta "libre" el día j
- $V_{i,j} = 1$: indica que la enfermera i esta asignada para el turno "larga" en el día j
- $V_{i,j} = 2$: indica que la enfermera i esta asignada para el turno "noche" en el día j

Una nueva variable denominada $B_{i,j}$, entregará los valores correspondientes a las enfermeras jefes. Su dominio será el siguiente:

- $B_{i,j} = 0$: indica que la enfermera jefe i esta "libre" el día j
- $B_{i,j} = 1$: indica que la enfermera jefe i esta asignada para el turno "larga" en el día j
- $B_{i,j} = 2$: indica que la enfermera jefe i esta asignada para el turno "noche" en el día j

Puesto que, por cada bloque ocho enfermeras tradicionales hay dos enfermeras jefes, en la variable $B_{i,j}$, el índice i recorrerá el bloque desde $Jefes$ hasta $Filas$. Por ejemplo, con ocho enfermeras ($TotalEnf = 8$), i recorrerá el intervalo $[8, 8 + 8/4]$. Por su parte, el índice j, recorre todas las columnas de la matriz, correspondiente al total de días.

Se tiene que todas las variables del modelo, se definen matemáticamente como:

$[V_{1,1}, V_{1,2}, \ldots, V_{i,j}] \in [0, 2]$, *tal que* $i \in [1, TotalEnf]$, $j \in [1, 28]$

$[B_{1,1}, B_{1,2}, \ldots, B_{i,j}] \in [0, 2]$, *tal que* $i \in [Jefes, Filas]$, $j \in [1, 28]$

Restricciones

En la realidad ocurre el caso que del grupo total de enfermeras existe un grupo de enfermeras que son más experimentadas, gracias a los años de servicio u otras que poseen mayor grado de prioridad y por motivos personales o beneficios por buen desempeño, se les permite elegir por cual turno empezar su ciclo de trabajo. Por lo que se ha propuesto a partir del primer modelo que un número determinado de enfermeras tengan restricciones propias, dadas por el turno en que desean comenzar el sistema del cuarto turno.

- $Para\, i \in [0, NJefes - 1]$, $donde\, NJefes = TotalEnf/8$

  1. $V_{1+(i*8),1} = 1 \vee V_{1+(i*8),1} = 2$
  2. $V_{2+(i*8),1} = 0 \vee V_{2+(i*8),1} = 2$
  3. $V_{3+(i*8),1} = 0 \vee V_{3+(i*8),1} = 1$

  Restricción con las posibles selecciones de turnos por parte de enfermeras con prioridad

Las enfermeras jefes poseen otro sistema de trabajo, realizan los mismos turnos, pero no la misma secuencia del sistema del cuarto turno. Su sistema laboral se establece gracias al conjunto de restricciones que se muestran a continuación:

- En un determinado día, por cada bloque de ocho enfermeras tradicionales, trabaja sólo una enfermera jefe en turno larga o noche.

  Siendo B_j, las columnas que representan sólo posiciones de la sección de enfermeras jefes, se cumple que:

  $Para\, j \in [1, TotalDias]$

  4. $occurrences(0, B_j, 1)$

- Una enfermera jefe no debe trabajar más de tres turnos larga en una semana:

  $Para\, i \in [Jefes, Filas] \wedge j \in [1, TotalDias - 3]$

  5. $(B_{i,j} = 1 \wedge B_{i,j+1} = 1 \wedge B_{i,j+2} = 1) \Rightarrow B_{i,j+3} = 0$

- Las enfermeras jefes no poseen un sistema rígido de turnos, pero sí una cantidad de turnos que deben cumplir mensualmente acompañado de una serie de reglas. La cantidad se define por cuatro turnos largas cada siete días (16 largas en 28 días) o tres turnos noche cada siete días (12 noches en 28 días).

  $Para\, j \in ([1, 7] \wedge [8, 14] \wedge [15, 21] \wedge [22, 28])$, $se\, debe\, cumplir$:

  6. $occurrences(1, B_j, 4)$

  7. $occurrences(2, B_j, 3)$
     Restricción para cuatro turnos larga y tres turnos noche

Por un lado se han presentado las restricciones propias de cada tipo de enfermera, por otro lado se deben establecer ciertos criterios para relacionar ambos tipos de enfermeras, los cuales se explican a continuación:

- Primero, las enfermeras jefes no tienen relación con las enfermeras experimentadas, ya que no son las que requieren mayor atención, al dejar enfermeras con prioridades se debe entender para este modelo que el resto del personal, sí requiere supervisión, tal vez por no poseer antigüedad laboral o ser profesionales recién egresadas, como también pueden ser enfermeras en práctica profesional. Para ello se ha establecido que las enfermeras con prioridad se ubican en las tres primeras filas de cada bloque de enfermeras, mientras que las sin prioridad se ubican en las cinco filas restantes.

- Segundo, la enfermera jefe que le corresponda trabajar, sea de día o de noche, está dada por la cantidad menor de enfermeras sin prioridad de un determinado turno, siempre y cuando dicha cantidad no sea igual a cero (ver caso 1 y 2 de tabla 3). Debido a que la mínima cantidad de enfermeras es uno, la enfermera jefe debe realizar trabajos en conjunto con la única enfermera de ese bloque, con el fin de brindar apoyo en diversas labores.

- Tercero, si una cantidad de enfermeras en un turno es igual a cero (ver caso 3 y 4 tabla 3), la enfermera jefe debe acompañar a las enfermeras que están trabajando, ya que debe supervisar en todo momento a lo menos a una enfermera.

- Cuarto, si las cantidades de enfermeras trabajando son iguales (ver caso 5 tabla 3), ambas enfermeras jefes podrán llegar a una acuerdo de quien trabaja, siempre y cuando no viole una de las restricciones de su sistema laboral (número de restricciones anteriores).

CASOS	N^o ENF. TURNO LARGA (1)		N^o ENF. TURNO NOCHE (2)	TURNO ENFERMERA JEFE
1	$ocurrencia(1)$	$>$	$ocurrencia(2)$	2
2	$ocurrencia(1)$	$<$	$ocurrencia(2)$	1
3	$ocurrencia(1) > 0$	$\wedge$	$ocurrencia(2) = 0$	1
4	$ocurrencia(1) = 0$	$\wedge$	$ocurrencia(2) > 0$	2
5	$ocurrencia(1)$	$=$	$ocurrencia(2)$	$1 \vee 2$

Tabla 3: Criterio para asignación de enfermeras jefes

La tabla 4 muestra de forma gráfica los casos de la tabla 3, con un ejemplo de cada uno de los cinco criterios para la asignación de enfermeras jefes.

9.3. Codificación del NRP avanzado en ECL^{i}PS^e

En esta sección se explicará la codificación en el lenguaje ECL^{i}PS^e del modelo avanzado para la resolución del NRP. Donde se añadió un grupo de nuevas restricciones y variables al modelo para lograr que la asignación cumpla con los nuevos requerimientos agregados al problema. La figura 14, contiene el código con las librerías, el predicado principal y la declaración de $Filas$, $Jefes$ y $NJefes$.

Luego se crean las restricciones que genera la prioridad de las tres enfermeras, estas corresponden a las restricciones 1, 2 y 3 del modelo avanzado. El ciclo for recorre por bloques, con el fin de asignar los valores a las tres primeras enfermeras de cada grupo de

SIMBOLOGIA				
Caso 1 ⇒ Negro	Caso 2 ⇒ Gris	Caso 3 ⇒ Azul	Caso 4 ⇒ Verde	Caso 5 ⇒ Rojo

	DIAS																		
ENFERMERAS	1	2	3	4	5	6	7	8	9	...	20	21	22	23	24	25	26	27	28
1	2	0	0	1	2	0	0	1	2	...	1	2	0	0	1	2	0	0	1
2	2	0	0	1	2	0	0	1	2	...	1	2	0	0	1	2	0	0	1
3	0	0	1	2	0	0	1	2	0	...	2	0	0	1	2	0	0	1	2
4	1	2	0	0	1	2	0	0	1	...	0	1	2	0	0	1	2	0	0
5	1	2	0	0	1	2	0	0	1	...	0	1	2	0	0	1	2	0	0
6	0	1	2	0	0	1	2	0	0	...	0	0	1	2	0	0	1	2	0
7	0	1	2	0	0	1	2	0	0	...	0	0	1	2	0	0	1	2	0
8	0	0	1	2	0	0	1	2	0	...	2	0	0	1	2	0	0	1	2

	DIAS																		
ENF JEFES	1	2	3	4	5	6	7	8	9	...	20	21	22	23	24	25	26	27	28
1	1	0	1	0	1	0	1	0	1	...	0	1	1	1	0	1	0	1	0
2	0	2	0	2	0	2	0	2	0	...	2	0	0	0	2	0	2	0	2

	DIAS																		
ENFERMERAS	1	2	3	4	5	6	7	8	9	...	20	21	22	23	24	25	26	27	28
9	2	0	0	1	2	0	0	1	2	...	1	2	0	0	1	2	0	0	1
10	0	1	2	0	0	1	2	0	0	...	0	0	1	2	0	0	1	2	0
11	1	2	0	0	1	2	0	0	1	...	0	1	2	0	0	1	2	0	0
12	2	0	0	1	2	0	0	1	2	...	1	2	0	0	1	2	0	0	1
13	1	2	0	0	1	2	0	0	1	...	0	1	2	0	0	1	2	0	0
14	0	1	2	0	0	1	2	0	0	...	0	0	1	2	0	0	1	2	0
15	0	0	1	2	0	0	1	2	0	...	2	0	0	1	2	0	0	1	2
16	0	0	1	2	0	0	1	2	0	...	2	0	0	1	2	0	0	1	2

	DIAS																		
ENF JEFES	1	2	3	4	5	6	7	8	9	...	20	21	22	23	24	25	26	27	28
3	1	1	0	1	1	0	0	1	1	...	1	1	1	0	1	0	1	0	1
4	0	0	2	0	0	2	2	0	0	...	0	0	0	2	0	2	0	2	0

Tabla 4: Casos para la asignación de enfermeras jefes

```
1 :-lib(ic).
2 :-lib(ic_global).
3
4 nrp(Matriz,TotalEnf,TotalDias) :-
5   Filas is TotalEnf+TotalEnf//4,
6   Jefes is 1+TotalEnf,
7   NJefes is TotalEnf//8,
```

Figura 14: Variables modelo avanzado NRP en ECL^{i}PS^e

8, dichos valores representan las prioridades de cada una de las tres enfermeras. En la figura 15, se puede apreciar como la primera enfermera tiene como prioridad empezar su ciclo con un turno larga o turno noche, la segunda prefiere comenzar con un día libre o de noche. Finalmente, la tercera puede empezar con un día libre o turno larga.

```
1 (for(I,0,NJefes-1),param(Matriz) do
2  Matriz[1+(I*8),1]#=1 or Matriz[1+(I*8),1]#=2,
3  Matriz[2+(I*8),1]#=0 or Matriz[1+(I*8),1]#=2,
4  Matriz[3+(I*8),1]#=0 or Matriz[1+(I*8),1]#=1
5 ),
```

Figura 15: Restricciones prioridad de enfermeras

La siguiente restricción, correspondiente al número 4 del modelo, valida la ocurrencia de una enfermera jefe por día, para un bloque de ocho enfermeras de turno. Al haber un jefe por día implica, independiente que sea larga o noche, que una de las dos debe tener día libre (no deben trabajar ambas).

Como es habitual en este modelo avanzado, la matriz es recorrida por bloques con un primer ciclo *for*. Luego se define la variable *Columna* con las dos enfermeras jefes. Para validar la restricción, se realiza *occurrences*, indicando que deba existir un turno libre cada dos enfermeras jefes por día. La figura 16 enseña el código fuente.

```
1  (for(I,0,NJefes-1),param(Matriz,TotalEnf,TotalDias) do
2   (for(J,1,TotalDias), param(Matriz, TotalEnf,I) do
3    Columna is Matriz[(TotalEnf+(I*2)+1)..(TotalEnf+(I*2)+2),J],
4    occurrences(0,Columna,1)
5   )
6  ),
```

Figura 16: Restricciones ocurrencia de una enfermera jefe por día

En seguida se declara la restricción 5 del modelo, utilizada para validar la cantidad de turnos larga de las enfermeras jefes, evitando la slecuencia de más de tres turnos larga consecutivos. El primer ciclo *for* que recorre las tres restricciones, se encarga de recorrer sólo las filas de enfermeras jefe, comenzando por $Jefes$ hasta $Filas$. En la figura 17, se muestran las restricciones que debe cumplir para satisfacer la secuencia.

En la figura 18, el fragmento de código implementa las restricciones 6 y 7 del modelo. Un primer ciclo *for*, recorre la matriz de enfermeras jefes en bloques de siete elementos. Por cada bloque visitado, dos ciclos *for* restringen la ocurrencia de turnos de día y noche mediante el predicado *occurrences*. Por ejemplo, en $occurrences(1, Fila, 4)$, se restringe que semanalmente ($Fila$) sólo puedan existir cuatro turnos de día (representados con el valor 1).

```
(for(I,Jefes,Filas),param(TotalDias,Matriz) do
 (for(J,1,TotalDias-3), param(I,Matriz) do
  (Matriz[I,J]#=1 and Matriz[I,J+1]#=1 and Matriz[I,J+2]#=1)
   => Matriz[I,J+3]#=0
  )
 ),
```

Figura 17: Restricciones para evitar tres turnos larga consecutivos

```
(for(H,1,22,7), param(Matriz,Filas,Jefes) do
 (for(J,Jefes,Filas,2), param(Matriz,H) do
  Fila is Matriz[J,H..H+6],
  occurrences(1,Fila,4)
 ),
 (for(J,Jefes+1,Filas,2), param(Matriz,H) do
  Fila is Matriz[J,H..H+6],
  occurrences(2,Fila,3)
 )
),
```

Figura 18: Restricciones de ocurrencia de turno larga y noche por cada semana

En las figuras 19, 20, 21 y 22, se muestra la codificación para cambiar los turnos de las enfermeras jefes en función de las otras enfermeras. El criterio empleado corresponde a los distintos casos de la tabla 3.

La figura 19 comienza con un primer ciclo *for* que recorrerá las filas de las enfermeras jefes. Un segundo ciclo *for* anidado, establece que por cada día se asignará una enfermera jefa con turno noche en la columna respectiva. Lo anterior sucede, si la ocurrencia de turnos de enfermeras de noche es menor que la cantidad de turnos de día, además si existe por lo menos un turno de día ($occurrences(2, Columna)\# < occurrences(1, Columna)\, and\, occurrences(1, Columna)\#\backslash = 0$).

```
(for(I,0,NJefes-1),param(Matriz,TotalDias,TotalEnf) do

 (for(J,1,TotalDias), param(Matriz,I,TotalEnf) do
  Columna is Matriz[(4+(I*8))..(8+(I*8)),J],
  ((occurrences(2,Columna)#<occurrences(1,Columna))
  and occurrences(2,Columna)#\=0) => Matriz[TotalEnf+(I*2)+2,J]#=2
 ),
```

Figura 19: Sección 1 asignación de enfermeras jefes

En la figura 20, se muestra un caso similar a la figura 19. Un ciclo *for* se encarga de recorrer la sección de enfermeras, rescatando cada día en la variable *Columna*. Si la

ocurrencia de turnos de enfermeras de noche en $Columna$ supera a la cantidad de turnos de día, además si existe por lo menos un turno de día ($occurrences(2, Columna)\# > occurrences(1, Columna)\, and\, occurrences(1, Columna)\#\backslash = 0$), asignará una enfermera jefe con turno larga.

```
1   (for(J,1,TotalDias), param(Matriz,I,TotalEnf) do
2    Columna is Matriz[(4+(I*8))..(8+(I*8)),J],
3    ((occurrences(2,Columna)#>occurrences(1,Columna))
4    and occurrences(1,Columna)#\=0) => Matriz[TotalEnf+(I*2)+1,J]#=1
5   ),
```

Figura 20: Sección 2 asignación de enfermeras jefes

En la figura 21, la codificación dentro del ciclo for asigna un turno de día para los casos donde el número de turnos de larga sea mayor que cero y no existan enfermeras con turnos de noche. La figura 22, sigue la misma lógica que la figura 21, asigna un turno de noche, para los casos donde el número de turnos noche sea mayor que cero y no existan enfermeras con turno larga.

```
1   (for(J,1,TotalDias), param(Matriz,I,TotalEnf) do
2    Columna is Matriz[(4+(I*8))..(8+(I*8)),J],
3    (occurrences(1,Columna)#>0
4    and occurrences(2,Columna)#=0) => Matriz[TotalEnf+(I*2)+1,J]#=1
5   ),
```

Figura 21: Sección 3 asignación de enfermeras jefes

```
1   (for(J,1,TotalDias), param(Matriz,I,TotalEnf) do
2    Columna is Matriz[(4+(I*8))..(8+(I*8)),J],
3    (occurrences(2,Columna)#>0
4    and occurrences(1,Columna)#=0) => Matriz[TotalEnf+(I*2)+2,J]#=2
5   )
6  ),
```

Figura 22: Sección 4 asignación de enfermeras jefes

El predicado principal `nrp`($Matriz, TotalEnf, TotalDias$) continúa con la llamada al predicado `labeling` encargado de encontrar la solución que satisfaga todas las restricciones. Por último, termina con `print_board`($Matriz, TotalEnf, Filas, TotalDias, NJefes$), que muestra los valores encontrados para la variable $Matriz$ (ver figura 24).

En la figura 23 se muestra el código fuente del predicado `print_board`. El primer for se encarga de recorrer la matriz que contiene ambos tipos de enfermeras, se desplaza

por bloques (ocho enfermeras de turno más dos enfermeras jefes). De acuerdo al índice del ciclo, ambos *for* en su interior podrán interactuar, el primero recorre las filas de enfermeras de turno (desde la fila uno hasta el total de dichas enfermeras), al recorrer ocho termina su función temporalmente para dar paso al segundo *for*, que desplegará de a dos enfermeras jefes (desde $TotalEnf + 1$ hasta $Filas$). El ciclo se repite de acuerdo a la cantidad de bloques existentes.

```
1  print_board(Matriz, TotalEnf, Filas, TotalDias, NJefes) :-
2   dim(Matriz, [Filas, TotalDias]),
3
4   (for(H,1,NJefes), param(Matriz, TotalDias, TotalEnf) do
5    (for(I,(H-1)*8+1,8*H), param(Matriz, TotalDias) do
6     (for(J,1,TotalDias), param(Matriz, I) do
7      X is Matriz[I,J],
8      printf(" %d", [X])
9     ), nl
10   ),nl,
11
12   (for(I,TotalEnf+(H-1)*2+1,TotalEnf+H*2), param(Matriz, TotalDias) do
13    (for(J,1,TotalDias), param(Matriz, I) do
14     X is Matriz[I,J],
15     printf(" %d", [X])
16     ), nl
17    ), nl
18  ).
```

Figura 23: Impresión por pantalla de resultados del NRP

9.4. Compilación y ejecución del NRP en ECL^{i}PS^e

La compilación se realiza en TkECLiPSe 6.0. En la figura 24 se muestra el resultado de la ejecución del código fuente. En este caso el código compilado se ejecuta con el predicado `nrp`$(M, 16, 28)$. Donde M será la variable que muestra el resultado, 16 corresponde al número de enfermeras y 28 corresponde al número total de días de la asignación.

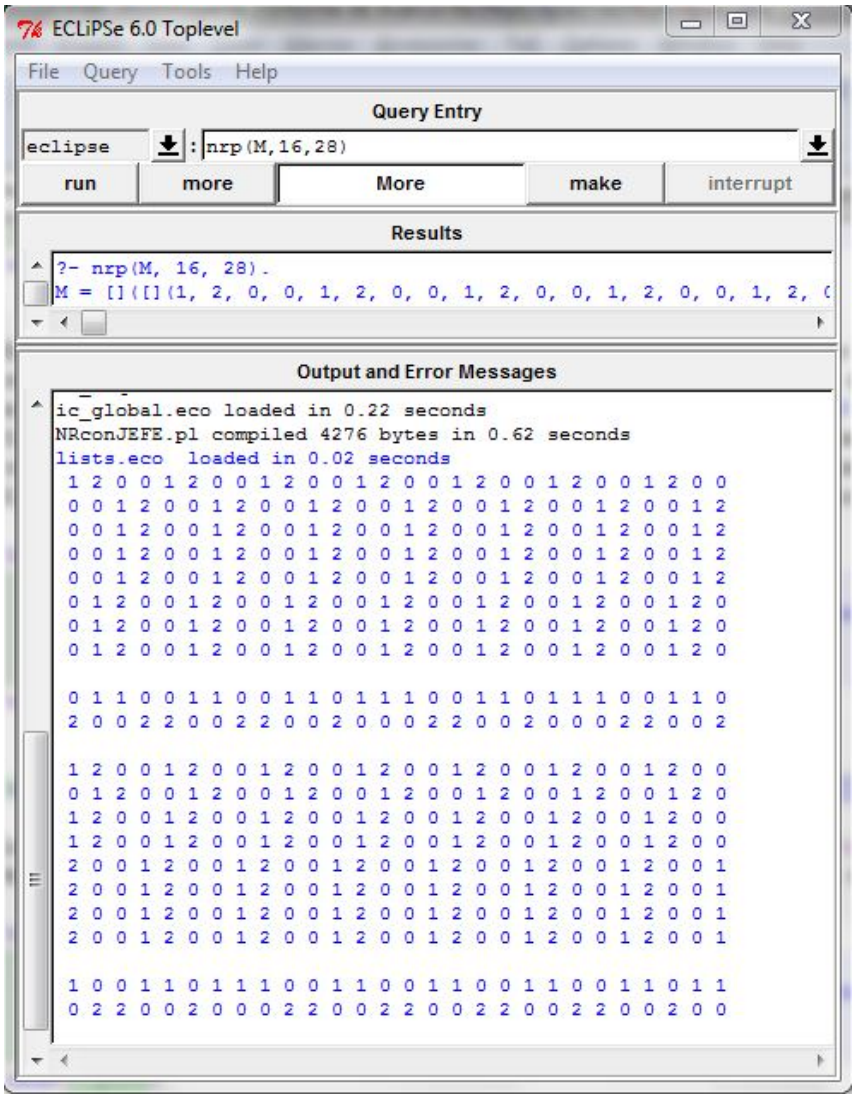

Figura 24: Resultado ejecución del NRP en TkECLiPSe

10. Pruebas

Las pruebas consisten en la comparación de distintas heurísticas, las cuales serán evaluadas con la medición de los tiempos en encontrar la primera solución y la cantidad de backtracks utilizados.

10.1. Heurísticas

A continuación se muestra la codificación realizada para la resolución del NRP, aplicando distintas heurísticas. El predicado utilizado para invocar a la distintas heurísticas, se denomina `estrategia`, el cual contiene los siguientes cuatro argumentos:

- Estrategia de selección de variable
- Estrategia de selección de valor
- La variable $Matriz$
- Contador de backtracks

Por ejemplo, para evaluar la estrategia que elige la variable con el menor dominio y el menor valor del dominio, que se muestra en la figura 25, se debe utilizar `estrategia`$(ff, min, Matriz, BT)$, donde ff corresponde a $first\_fail$, min representa el valor mínimo de la selección de valor, $Matriz$ contiene todas las variables del problema y BT retorna el número de backtracks de la búsqueda.

Las figuras 25, 26, 27, 28 y 29, muestran la codificación de las heurísticas. Se observa que todas utilizan el predicado $search/6$, proporcionado por la librería ic_search de ECL^{i}PS^e. $search/6$ provee un amplio rango de métodos de búsqueda, que permiten establecer estrategias de selección de variable y valor. En el ámbito de la selección de valor se utilizan los siguientes métodos predefinidos:

- input_order: se selecciona la primera entrada en la lista.
- first_fail: se selecciona la variable con menor dominio.
- anti_first_fail: se selecciona la variable con mayor dominio.
- most_constrained: es un método heurístico que implica trabajar con aquellas variables del problema que tienen el menor número de opciones válidas (menor dominio), esto es, elegir primero aquellas opciones que tienen el mayor número de conflictos, para que las decisiones siguientes tengan un rango menor de posibilidades. En caso de existir igualdad en el tamaño del dominio, la elección se realizará por medio de la variable que tenga mayor cantidad de restricciones asociadas.

Otra estrategia que forma parte de la evaluación, corresponde a $round-robin$, la cual selecciona todos los elementos de un grupo de forma equitativa y con un orden racional, por lo general comenzando con el primer elemento de la lista hasta el último y luego

comenzando nuevamente con el primer elemento. La codificación de $round - robin$ no es posible llevarla a cabo directamente con el predicado $search/6$, es por ello que requiere una codificación diferente a las anteriores. En el anexo C, se muestra su implementación.

Referido a la selección de valor, se utilizan los métodos predefinidos *indomain_min*, *indomain_middle* y *indomain_max*, para seleccionar el mínimo, medio y máximo valor de la lista respectivamente.

```
1 estrategia(ff,min, AllVars, BT) :-
2  search(AllVars,0,first_fail,indomain_min,complete,[backtrack(BT)]).
3
4 estrategia(ff,middle, AllVars, BT) :-
5  search(AllVars,0,first_fail,indomain_middle,complete,[backtrack(BT)]).
6
7 estrategia(ff,max, AllVars, BT) :-
8  search(AllVars,0,first_fail,indomain_max,complete,[backtrack(BT)]).
```

Figura 25: Codificación estrategia first-fail

```
1 estrategia(aff,min, AllVars, BT) :-
2  search(AllVars,0,anti_first_fail,indomain_min,complete,
3  [backtrack(BT)]).
4
5 estrategia(aff,middle, AllVars, BT) :-
6  search(AllVars,0,anti_first_fail,indomain_middle,complete,
7   [backtrack(BT)]).
8
9 estrategia(aff,max, AllVars, BT) :-
10  search(AllVars,0,anti_first_fail,indomain_max,complete,
11  [backtrack(BT)]).
```

Figura 26: Codificación estrategia anti-first-fail

10.2. Resultados

El problema modelado fue implementado y resuelto en la plataforma ECL^{i}PS^e, con las estrategias expuestas anteriormente. Se utilizó un computador con las siguientes características:

- Procesador Intel Core 2 Duo, 2,00 GHZ
- Sistema Operativo Microsoft Windows 7 Ultimate (32 bits)
- 3 GB Memoria RAM, DDR2-800, 400 MHZ

```
1  estrategia(io,min, AllVars, BT) :-
2   search(AllVars, 0, input_order, indomain_min, complete,
3   [backtrack(BT)]).
4
5  estrategia(io,middle, AllVars, BT) :-
6   search(AllVars, 0, input_order, indomain_middle, complete,
7   [backtrack(BT)]).
8
9  estrategia(io,max, AllVars, BT) :-
10  search(AllVars, 0, input_order, indomain_max, complete,
11  [backtrack(BT)]).
```

Figura 27: Codificación estrategia input-order

```
1  estrategia(mc,min, AllVars, BT) :-
2   search(AllVars, 0, most_constrained, indomain_min, complete,
3   [backtrack(BT)]).
4
5  estrategia(mc,middle, AllVars, BT) :-
6   search(AllVars, 0, most_constrained, indomain_middle, complete,
7   [backtrack(BT)]).
8
9  estrategia(mc,max, AllVars, BT) :-
10  search(AllVars, 0, most_constrained, indomain_max, complete,
11  [backtrack(BT)]).
```

Figura 28: Codificación estrategia most-constrained

```
1  estrategia(lb,min, AllVars, BT) :-
2   search(AllVars, 0, input_order, indomain, complete, [backtrack(BT)]).
```

Figura 29: Codificación estrategia por defecto de ECL^{i}PS^e (labeling)

Cada ejecución del problema tuvo un límite de tiempo de 60 minutos, los resultados no encontrados son indicados con la palabra "timeout". Asimismo, las pruebas consideran un tamaño máximo de utilización de Memoria RAM de 1363 MB (límite permitido por software TkEclipse), los desbordamientos de memoria se indican con la palabra "overflow".

Las pruebas realizadas permitieron evaluar el desempeño de las estrategias de enumeración, basándose en dos indicadores de rendimiento, el tiempo y el número de backtracks requeridos para resolver el NRP.

10.2.1. Tiempos de resolución

En la tabla 5 y tabla 6 se muestran los resultados de los tiempos obtenidos, medidos en segundos.

TIEMPO (segundos)								
	Número de enfermeras							
Estrategia	**8**	**16**	**24**	**32**	**40**	**48**	**56**	**64**
first_fail+min	0,2184	0,4524	0,7176	1,0140	1,3260	1,6692	2,1528	2,4960
first_fail+middle	0,0624	0,1092	0,1872	0,2340	0,3120	0,4056	0,5928	0,6864
first_fail+max	0,2340	0,4836	0,7644	1,0764	1,3884	1,7472	2,2308	2,5061
anti_first_fail+min	891,0309	timeout	timeout	timeout	timeout	timeout	timeout	timeout
anti_first_fail+middle	0,6240	137,5928	timeout	timeout	timeout	timeout	timeout	timeout
anti_first_fail+max	45,5054	timeout	timeout	timeout	timeout	timeout	timeout	timeout
input_order+min	0,2184	0,4524	0,6708	0,8892	1,1076	1,3416	1,5600	1,7784
input_order+middle	0,0468	0,1092	0,1560	0,2028	0,2652	0,3120	0,3588	0,4212
input_order+max	0,5148	1,0296	1,5288	2,0436	2,5584	3,0732	3,5880	4,0872
most_constrained+min	0,0936	0,2496	0,4992	0,9048	1,3416	1,8408	2,4180	3,0888
most_constrained+middle	0,0936	0,2496	0,4836	0,8736	1,2948	1,7628	2,3244	2,9172
most_constrained+max	0,1092	0,2808	0,6396	0,9672	1,4196	1,9656	2,5584	3,1980
round_robin+min	0,0468	0,1092	0,1560	0,2028	0,2652	0,3120	0,3744	0,4212
round_robin+middle	0,0468	0,1092	0,1560	0,2028	0,2496	0,3120	0,3588	0,4212
round_robin+max	0,3900	0,7800	1,170	1,5600	1,9656	2,3556	2,7456	3,1200
labeling	0,2184	0,4368	0,6708	0,8892	1,1076	1,3260	1,5444	1,7628

Tabla 5: Sección 1 cantidad de segundos para la resolución del NRP

10.2.2. Backtracks

El indicador del número de backtracks muestra la cantidad de malas decisiones hechas durante la búsqueda de la solución, es decir, los cálculos y decisiones ejecutadas sin llegar a una solución [5]. Los resultados de la medición de dicho parámetro se muestran en la tabla 7 y tabla 8.

10.3. Análisis de resultados

Antes de cualquier análisis es necesario entender que el trabajo que realiza un *solver* para encontrar la primera solución se ve influenciado en gran medida por la técnica de filtrado que este utiliza al seleccionar los valores. Para el caso concreto de ECL^{i}PS^e, las técnicas utilizadas privilegiarán a los valores pequeños del dominio mientras que a medida

TIEMPO (segundos)				
	Número de enfermeras			
Estrategia	**128**	**256**	**512**	**1024**
first_fail+min	6,6300	19,8277	66,5968	252,1756
first_fail+middle	1,8096	5,2104	16,8637	overflow
first_fail+max	6,7548	19,7029	65,1772	overflow
anti_first_fail+min	timeout	timeout	timeout	timeout
anti_first_fail+middle	timeout	timeout	timeout	timeout
anti_first_fail+max	timeout	timeout	timeout	timeout
input_order+min	3,6192	7,6596	15,0228	30,8881
input_order+middle	0,9360	2,3556	4,5084	9,7656
input_order+max	8,0808	16,5205	32,7446	66,4096
most_constrained+min	11,2320	42,6974	166,1254	overflow
most_constrained+middle	10,6392	40,4042	156,8434	overflow
most_constrained+max	11,5284	43,2902	167,3422	overflow
round_robin+min	0,9672	2,4024	4,6176	10,218
round_robin+middle	0,9360	2,2932	4,4772	9,7188
round_robin+max	6,2088	12,8388	25,3657	51,6207
labeling	3,5568	7,5816	14,8980	30,5761

Tabla 6: Sección 2 cantidad de segundos para la resolución del NRP

BACKTRACKS								
	Número de enfermeras							
Estrategia	**8**	**16**	**24**	**32**	**40**	**48**	**56**	**64**
first_fail+min	13	26	39	52	65	78	91	104
first_fail+middle	0	0	0	0	0	0	0	0
first_fail+max	12	24	36	48	60	72	84	96
anti_first_fail+min	301531	timeout	timeout	timeout	timeout	timeout	timeout	timeout
anti_first_fail+middle	70	15746	timeout	timeout	timeout	timeout	timeout	timeout
anti_first_fail+max	5563	timeout	timeout	timeout	timeout	timeout	timeout	timeout
input_order+min	12	24	36	48	60	72	84	96
input_order+middle	0	0	0	0	0	0	0	0
input_order+max	32	64	96	128	160	192	224	256
most_constrained+min	0	0	0	0	0	0	0	0
most_constrained+middle	0	0	0	0	0	0	0	0
most_constrained+max	0	0	0	0	0	0	0	0
round_robin+min	0	0	0	0	0	0	0	0
round_robin+middle	0	0	0	0	0	0	0	0
round_robin+max	24	48	72	96	120	144	168	192
labeling	12	24	36	48	60	72	84	96

Tabla 7: Sección 1 número de backtracks para la resolución del NRP

BACKTRACKS				
	Número de enfermeras			
Estrategia	**128**	**256**	**512**	**1024**
first_fail+min	208	416	832	1664
first_fail+middle	0	0	0	overflow
first_fail+max	192	384	768	overflow
anti_first_fail+min	timeout	timeout	timeout	timeout
anti_first_fail+middle	timeout	timeout	timeout	timeout
anti_first_fail+max	timeout	timeout	timeout	timeout
input_order+min	192	384	768	1536
input_order+middle	0	0	0	0
input_order+max	512	1024	2048	4096
most_constrained+min	0	0	0	overflow
most_constrained+middle	0	0	0	overflow
most_constrained+max	0	0	0	overflow
round_robin+min	0	0	0	0
round_robin+middle	0	0	0	0
round_robin+max	384	778	1536	3072
labeling	192	384	768	1536

Tabla 8: Sección 2 número de backtracks para la resolución del NRP

que se aproximan al valor máximo del dominio, el rendimiento será decreciente. En base a lo anterior, lo esperado para el NRP sería que el valor *min* del dominio presentara un mejor filtrado que el valor *middle* y a su vez este último mejor resultado que el valor *max*, no obstante, las estrategias que se utilizaron y las restricciones que instancian a los valores inciden claramente en el tiempo y los backtracks (ver tablas de resultados). Algunas observaciones relacionadas con este comportamiento se explican a continuación.

- En la totalidad de los casos de prueba los resultados muestran que ya sea para un bajo o un alto número de enfermeras, la estrategia de enumeración que obtiene el mejor rendimiento, ya sea en tiempo y número de backtracks es $round - robin$ con la selección del valor medio (*middle*).

- Para todas las estrategias de selección de variable evaluadas, la selección de valor *middle* presenta mejor tiempo que *max*, si bien tienen similares restricciones, el hecho de realizar un mejor filtrado hace posible que la solución sea encontrada de forma más rápida. A su vez, el mismo criterio de filtrado ocasiona que la cantidad de malas decisiones (backtracks) por parte del valor *max* sea igual o superior al valor *middle*.

- La selección de valor *min* presenta resultados más tardíos en comparación al valor *middle*. Esto podría explicarse, por el hecho de que el *solver* requiere mayor tiempo en filtrar las combinaciones erróneas de enfermeras de turno libre, que turno larga. Por ejemplo, al observar la figura 24 de un total de 16 enfermeras tradicionales (8 tienen día libre , 4 trabajan de día y 4 de noche), el *solver* requiere mayor tiempo en filtrar asignaciones equívocas de ocho enfermeras de turno libre que de cuatro enfermeras de turno larga.

- El peor resultado fue obtenido por $anti - first - fail$, que al ser lo contario de $first - fail$ presenta tiempos desmedidos en obtener la primera solución, ya que seleccionará para cada variable los dominios más amplios, los cuales poseen una mayor cantidad de valores disponibles, por ende, una menor probabilidad de fallar.

  En relación a las demás estrategias existen notorias diferencias. Por una parte, los tiempos son desmedidos al punto que consultar, por ejemplo, para 16 (min y max) y 24 enfermeras el tiempo máximo de espera es sobrepasado con creces (ver tabla 5). Por otra parte, la cantidad de malas decisiones en las tres selecciones de valor es alcanzando cifras exageradas como 301531 backtracks (ver tabla 7).

11. Conclusiones

El largo trabajo de encontrar un modelo final del NRP para cumplir los objetivos se ha dado por finalizado, no obstante, hubo variadas dificultades, tareas necesarias y problemas que fueron surgiendo a medida que se involucraron nuevas características al problema, de ahí la importancia de separar el desarrollo en etapas fue clave.

En el caso particular de este proyecto, se ha modelado el NRP en dos fases, cada uno con distintos grados de dificultad. Por un lado, el modelo inicial responde a la planificación real de Clínica Valparaíso, que fue modelado de forma breve, ya que la relación entre la cantidad de variables, constantes y restricciones no presentó mayor complejidad. Se habla de baja complejidad en comparación con problemas similares de otros centros médicos, los cuales incluyen por ejemplo la jornada de tarde, otros que no siguen un patrón de rotación de cuatro días e incluso duración de la jornadas con menos o más horas (seis u ocho horas). Por otro lado, el segundo modelo desarrollado tenía como finalidad aumentar la dificultad del modelo preliminar incrementando la cantidad de variables (enfermera jefe) y a su vez la cantidad de restricciones en el modelo

Para el correcto desarrollo del proyecto se especifican cuatro etapas, la primera consistió en adquirir los conocimientos necesarios para entender un CSP (fundamentos, formalización, algoritmos y heurísticas), para facilitar el trabajo el ejemplo del $N-reinas$ fue de gran ayuda, sobre todo para entender los algoritmos.

La segunda etapa, consistió en una visita a Clínica Valparaíso con el propósito de rescatar y comprender el funcionamiento interno de un centro asistencial real. Se obtuvo terminología como el sistema del cuarto turno, el cual consiste en un periodo rotativo de cuatro días donde los turnos siguen un patrón formado por uno de día, uno de noche y dos días libres, cada turno consta de doce horas continuas. Producto de dicha visita, en el documento [24] se presenta un modelado inicial y resultados de algunos casos de prueba realizados en ECL^{i}PS^e.

Ya con las variables del problema identificadas, se crean las diversas restricciones en base a lo conversado con la enfermera jefe de la Clínica (que entregó la información recién mencionada) y se establecen los dominios respectivos para cada restricción. En una primera instancia, se modelan las restricciones para luego en una segunda, transcribirlas a ECL^{i}PS^e, el cual fue escogido dentro de un listado expuesto en este documento. La elección de este *solver* se debe al manejo que ya se tenía sobre algunos predicados y nomenclatura que posee el lenguaje como la sintaxis y el llamado a otros predicados, que en su conjunto proporcionaban las herramientas necesarias para un desarrollo que cumpliera con los objetivos planteados.

En una tercera etapa, se crea una versión avanzada que si bien, se desprende de la planificación real del caso de estudio, agrega dificultad al modelado con la inclusión de un número mayor de restricciones al momento de combinar las variables (enfermeras jefes con las enfermeras tradicionales del modelo preliminar), tales como, la jornada de trabajo semanal de la enfermera jefe de noche con la de día, la cantidad de turnos máximos de una enfermera, o la supervisión que deben tener dichas enfermeras con aquellas sin experiencia o alumnas practicantes.

Asimismo, otra situación complicada se originó al dividir la matriz en dos sub matrices

(enfermeras jefes y enfermeras tradicionales) y a su vez relacionar los índices de éstas, como fue por ejemplo en los cálculos de ocurrencias. En primera instancia se propuso trabajar con dos matrices, pero presentó un problema importante, ya que la cantidad de variables aumentaría de manera considerable y alejaría el hecho de crear un modelo simplificado. En conclusión, se optó por unir ambas sub matrices con el fin de reducir las variables, lo cual facilita en gran medida el desarrollo.

La última etapa, consistió en la aplicación de las heurísticas al modelo. La codificación y aplicación de las estrategias de enumeración permitió comparar y elegir la estrategia que se comporta de mejor manera. La elección de la estrategia de mejor rendimiento surgió de la observación de dos parámetros, el tiempo y el número de backtracks.

Lo que respecta al análisis de resultados, se espera un tiempo mínimo y lineal a medida que aumentan el número de enfermeras. Las que responden a este requerimiento son $input-order$, $round-robin$ y $labeling$, mientras que las otras incrementan su tiempo de manera exponencial, sobrepasando el tiempo máximo de espera o provocando overflow. Con respecto a los backtracks, la diferencias en el número de malas decisiones hechas por el *solver* antes de llegar a la solución, se espera que sean cercanas a cero. En definitiva, la heurística que cumple estos requisitos es la selección de variable *round robin* con la selección del valor medio (*middle*), que en todos los casos de prueba fue la que entregó los mejores tiempos de resolución y menor número de backtracks.

Después del análisis del comportamiento de ECL^{i}PS^e en relación a los resultados, se infiere que la razón por la cual ciertas estrategias entregan mejores tiempos de respuesta, en parte se debe a que el *solver* realiza un filtrado más eficiente con los valores mínimos, dicho rendimiento disminuye a medida que se eligen valores mayores (el valor 0 es mejor que el valor 1, este último mejor que el valor máximo 2). A su vez, para este modelo en particular, los tiempos con estrategias de selección medio, son comparativamente mejores que la selección de dominios mínimos, debido a las diferencias de restricciones involucradas en los turnos libre y de día que debe asignar ECL^{i}PS^e.

En cuanto a futuras investigaciones, el modelo resuelto en el presente documento cumple con la estructura de turnos extraída de un caso real. No obstante, la necesidad de obtener un modelo genérico es importante, pero implicaría una investigación de otros sistemas, ya que no todos los centros médicos trabajan de la misma manera que el caso de estudio de Clínica Valparaíso. Entre algunas modificaciones que pueden ser desarrolladas para perfeccionar el modelo se encuentran variar la cantidad de personal entre un área y otra; cambiar la cantidad de personal, incluyendo paramédicos, asistentes; implementar otros tipos de calendarización (trimestral, semestral, anual) para distintas especialidades; añadir nuevos sistema de turnos con distintos horarios y periodos.

12. Referencias

[1] K.R. Apt. *Principles of Constraint Programming.* Cambridge Press, 2003.

[2] KR Apt and MG Wallace. Constraint Logic Programming Using ECL^{i}PS^e. *Cambridge Univerity Press*, 2007.

[3] F. Barber and M. Salido. Introducción a la programación con restricciones. *Revista Iberoamericana de Inteligencia Artificial*, páginas 9–10, 2003.

[4] N. Beldiceanu and E. Contjean. Introducing global constraints in chip, mathematical and computer modelling. 1994.

[5] Eric Monfroy Broderick Crawford, Carlos Castro and Nibaldo Rodríguez. Solving constraint satisfaction puzzles with constraint programming. *Congreso de Inteligencia Computacional Aplicada, CIC 2009, Buenos Aires, Argentina*, 2009.

[6] Chan S.H.C. Lam G.P.S. Tsang F.M.F. Wong J. Chun, A.H.W. and Yeung. Nurse Rostering at the Hospital Authority of Hong Kong. *Proc. of 17th National Conference on AAAI and 12th Conference on IAAI*, 2000.

[7] A. Colmerauer. Prolog II Reference Manual and Theoretical Model. 1982.

[8] D. Diaz and p. Codognet. The GNU Prolog System and its Implementation. páginas 728–732, 2000.

[9] A. Dresse. A Constraint Programming Library Dedicated to Timetabling. *Proceedings of the First ILOG Solver and Scheduler Users Conference.*, 1995.

[10] P. De Causmaecker E.K. Burke and G. Vanden Berghe. A hybrid tabu search algorithm for the nurse rostering problem. *Lecture Notes in Artificial Intelligence*, 1999.

[11] S. Petrovic G. Vanden Berghe E.K. Burke, P. De Causmaecker. Variable neighbourhood search for nurse rostering problems. *Metaheuristics International Conference Proceedings*, Julio 2001.

[12] Timothy Curtois Rong Qu E.K. Burke, P. De Causmaecker, Greet, and G. Vanden Berghe. A Time Pre-defined Variable Depth Search for Nurse Rostering. 2007.

[13] D. Frost. Algorithms and Heuristics for Constraint Satisfaction Problems. *Ph.D. Thesis, Universidad de California, Irvine, CA*, 1997.

[14] D. Frost and R. Dechter. Look-ahead value orderings for constraint satisfaction problems. *Universidad de California*, 1995.

[15] P. Van Hentenryck. The OPL Language. *The MIT Press*, 1999.

[16] J. Jaffar and J. L. Lassez. Constraint Logic Programming. *14th Annual ACM Symposium on Principles of Programming Languages (POPL)*, páginas 111–119, 1987.

[17] Patrice Boizumault Jean-Philippe Metivier and Samir Loudni. Solving nurse rostering problems using soft global constraints. *The 7th International Conference on the Practice and Theory of Automated Timetabling, PATAT 2008, Montreal*, 2009.

[18] S Kusumoto. Nurse Scheduling System Using ILOG Solver. *In Proceedings of the Second ILOG Solver and Scheduler Users Conference*, 1996.

[19] S. Novello M. Wallace and J. Schimpf. ECL^{i}PS^e: A Platform for Constraint Logic Programming. *Imperial College, London*, 1997.

[20] S. Martello and P. Toth. A Heuristic Approach to the Bus Driver Scheduling Problem. *European Journal of Operations Research*, 24(1):106–117, 1986.

[21] R. Becket S. Brand G. J. Duck N. Nethercote, P. J. Stuckey and G. Tack. MiniZinc: Towards A Standard CP Modelling Language. *Springer*, 2007.

[22] T. Frühwirth C. Gervet W. Harvey M. Meier S. Novello T. L. Provost J. Schimpf K. Shen P. Brisset, H. E. Sakkout and M. Wallace. ECL^{i}PS^e Constraint Library Manual.

[23] Berghe G.V. Petrovic, S. Comparison of algorithms for nurse rostering problems. *The 7th International Conference on the Practice and Theory of Automated Timetabling, PATAT 2008, Montreal*, 2008.

[24] Renzo Pizarro, Gianni Rivera, Ricardo Soto, Broderick Crawford, Carlos Castro, and Eric Monfroy. Constraint-Based Nurse Rostering for the Valparaíso Clinic Center in Chile. En *HCI (23)*, páginas 448–452, 2011.

[25] I. Meiri R. Dechter. Experimental evaluation of preprocessing algorithms for constraints satisfaction problems. *Artificial Intelligence*, 1994.

[26] K. Marriot R. Rafeh, M. J. García de la Banda and M. Wallace. From zinc to design model. 2007.

[27] E.S. Rosenbloom and Goertzen. Cyclic Nurse Scheduling. *European Journal of Operations*, 1987.

[28] J. Puchinger S. Brand, G. J. Duck and P. J. Stuckey. Flexible. Rule-Based Constraint Model Linearisation. *Springer*, páginas 68–83, 2008.

[29] A.B. Philips S. Minton, M.D. Johnston and P. Laird. A heuristic repair method for constraint-satisfaction and scheduling problems. *Artificial Intelligence*, 1992.

[30] C. Schulte and G. Tack. Views and Iterators for Generic Constraint Implmentations. *Springer*, páginas 118–132, 2006.

[31] Sistema de Koalog. http://www.koalog.com, Revisada por última vez el 17 de agosto de 2010.

[32] S.U. and D. Sitompul. A Heuristic Based Computerized Nurse Scheduling System. *Computers and Operations*, 1983.

Anexos

A: Código fuente resolución del NRP preliminar en ECL^{i}PS^e

```
:-lib(ic).
:-lib(ic_global).

nrp(Matriz,TotalEnf,TotalDias) :-
 dim(Matriz, [TotalEnf,TotalDias]),
 Matriz[1..TotalEnf,1..TotalDias] :: 0..2,

%--Secuencia del cuarto turno-----------------------------------

 (for(I,1,TotalEnf),param(TotalDias,Matriz) do
  (for(J,1,TotalDias-2), param(I,Matriz) do
   (Matriz[I,J]#=0 and Matriz[I,J+1]#=0) => Matriz[I,J+2]#=1 ,
  Matriz[I,J]#=2 => (Matriz[I,J+1]#=0 and Matriz[I,J+2]#=0)
   )
  ),

 (for(I,1,TotalEnf),param(TotalDias,Matriz) do
  (for(J,1,TotalDias-1), param(I,Matriz) do
   Matriz[I,J]#=1 => Matriz[I,J+1]#=2,
   Matriz[I,J]#=2 => Matriz[I,J+1]#=0
  )
 ),

%--Cantidad de enfermeras por día-------------------------------

 (for(K,1,TotalEnf-7,8),param(Matriz,TotalDias) do
  (for(J,1,TotalDias), param(Matriz,K) do
   Columna is Matriz[K..K+7,J],
   occurrences(0,Columna,4),
   occurrences(1,Columna,2),
   occurrences(2,Columna,2)
  )
 ),
 labeling(Matriz),
 print_board(Matriz,TotalEnf,TotalDias).

%--Muestra resultados-------------------------------------------

print_board(Matriz,TotalEnf,TotalDias) :-
 dim(Matriz, [TotalEnf,TotalDias]),
(for(I,1,TotalEnf), param(Matriz,TotalDias) do
 (for(J,1,TotalDias), param(Matriz,I) do
   X is Matriz[I,J],
   printf(" %d", [X])
  ), nl
 ), nl.
```

B: Código fuente resolución del NRP avanzado en ECL^{i}PS^e

```
:-lib(ic).
:-lib(ic_global).

nrp(Matriz,TotalEnf,TotalDias) :-
 Filas is TotalEnf+TotalEnf//4,
 Jefes is 1+TotalEnf,
 dim(Matriz, [Filas,TotalDias]),
 Matriz[1..Filas,1..TotalDias] :: 0..2,
 NJefes is TotalEnf//8,

%--Prioridad de enf-tradicionales-----------------------------

 (for(I,0,NJefes-1),param(Matriz) do
  Matriz[1+(I*8),1]#=1 or Matriz[1+(I*8),1]#=2,
  Matriz[2+(I*8),1]#=0 or Matriz[1+(I*8),1]#=2,
  Matriz[3+(I*8),1]#=0 or Matriz[1+(I*8),1]#=1
 ),

%--Modelo preliminar------------------------------------------

 (for(I,1,TotalEnf),param(TotalDias,Matriz) do
  (for(J,1,TotalDias-2), param(I,Matriz) do
  ( Matriz[I,J]#=0 and Matriz[I,J+1]#=0) => Matriz[I,J+2]#=1 ,
  Matriz[I,J]#=2 => (Matriz[I,J+1]#=0 and Matriz[I,J+2]#=0)
  )
 ),

 (for(I,1,TotalEnf),param(TotalDias,Matriz) do
  (for(J,1,TotalDias-1), param(I,Matriz) do
  Matriz[I,J]#=1 => Matriz[I,J+1]#=2,
  Matriz[I,J]#=2 => Matriz[I,J+1]#=0
  )
 ),

 (for(K,1,TotalEnf-7,8),param(Matriz,TotalDias) do
  (for(J,1,TotalDias), param(Matriz,K) do
  Columna is Matriz[K..K+7,J],
  occurrences(0,Columna,4),
  occurrences(1,Columna,2),
  occurrences(2,Columna,2)
  )
 ),

%--Restricciones enfermeras jefe-----------------------------

(for(I,Jefes,Filas),param(TotalDias,Matriz) do
  (for(J,1,TotalDias-3), param(I,Matriz) do
    ( Matriz[I,J]#=1 and Matriz[I,J+1]#=1 and Matriz[I,J+2]#=1)
    => Matriz[I,J+3]#=0
  )
```

```
),

 (for(I,0,NJefes-1),param(Matriz,TotalEnf,TotalDias) do
  (for(J,1,TotalDias), param(Matriz,  TotalEnf,I) do
  Columna is Matriz[(TotalEnf+(I*2)+1)..(TotalEnf+(I*2)+2),J],
  occurrences(0,Columna,1)
  )
 ),

%--Restricciones para cada semana-------------------------

(for(H,1,22,7), param(Matriz,Filas,Jefes) do
  (for(J,Jefes,Filas,2), param(Matriz,H) do
  Fila is Matriz[J,H..H+6],
  occurrences(1,Fila,4)
  ),
  (for(J,Jefes+1,Filas,2), param(Matriz,H) do
  Fila is Matriz[J,H..H+6],
  occurrences(2,Fila,3)
  )
 ),

 %--Criterio para relacionar enf-jefe con enf-tradicional-

 (for(I,0,NJefes-1),param(Matriz,TotalDias,TotalEnf) do

  (for(J,1,TotalDias), param(Matriz,I,TotalEnf) do
   Columna is Matriz[(4+(I*8))..(8+(I*8)),J],
   ((occurrences(2,Columna)#<occurrences(1,Columna))
   and occurrences(2,Columna)#\=0 ) =>  Matriz[TotalEnf+(I*2)+2,J]#=2
  ),

  (for(J,1,TotalDias), param(Matriz,I,TotalEnf) do
   Columna is Matriz[(4+(I*8))..(8+(I*8)),J],
   ((occurrences(2,Columna)#>occurrences(1,Columna))
   and occurrences(1,Columna)#\=0 ) =>  Matriz[TotalEnf+(I*2)+1,J]#=1
  ),

  (for(J,1,TotalDias), param(Matriz,I,TotalEnf) do
   Columna is Matriz[(4+(I*8))..(8+(I*8)),J],
   (occurrences(1,Columna)#>0 and occurrences(2,Columna)#=0)
    => Matriz[TotalEnf+(I*2)+1,J]#=1
  ),

  (for(J,1,TotalDias), param(Matriz,I,TotalEnf) do
   Columna is Matriz[(4+(I*8))..(8+(I*8)),J],
   (occurrences(2,Columna)#>0 and occurrences(1,Columna)#=0 )
   => Matriz[TotalEnf+(I*2)+2,J]#=2
   )
 ),

 labeling(Matriz),
```

```
 print_board(Matriz,TotalEnf,Filas,TotalDias,NJefes).


%--Muestra resultados-----------------------------------------

print_board(Matriz,TotalEnf,Filas,TotalDias,NJefes) :-
 dim(Matriz, [Filas,TotalDias]),
 (for(H,1,NJefes), param(Matriz,TotalDias,TotalEnf) do
  (for(I,(H-1)*8+1,8*H), param(Matriz,TotalDias) do
   (for(J,1,TotalDias), param(Matriz,I) do
    X is Matriz[I,J],
  printf(" %d", [X])
   ), nl
  ),nl,
  (for(I,TotalEnf+(H-1)*2+1,TotalEnf+H*2), param(Matriz,TotalDias) do
  (for(J,1,TotalDias), param(Matriz,I) do
   X is Matriz[I,J],
   printf(" %d", [X])
  ), nl
  ), nl
 ).
```

C: Código fuente estrategias de resolución en NRP avanzado de ECL^{i}PS^e

```
%-----------------------------------------------------------------------
%estrategia First fail + ordenamiento de valor (min,middle,max)
%-----------------------------------------------------------------------
estrategia(ff,min, AllVars, BT) :-
 search(AllVars, 0, first_fail, indomain_min, complete,
 [backtrack(BT)]).

estrategia(ff,middle, AllVars, BT) :-
 search(AllVars, 0, first_fail, indomain_middle, complete,
 [backtrack(BT)]).

estrategia(ff,max, AllVars, BT) :-
 search(AllVars, 0, first_fail, indomain_max, complete,
 [backtrack(BT)]).

%-----------------------------------------------------------------------
%estrategia Anti first fail + ordenamiento de valor (min,middle,max)
%-----------------------------------------------------------------------
estrategia(aff,min, AllVars, BT) :-
 search(AllVars, 0, anti_first_fail, indomain_min, complete,
 [backtrack(BT)]).

estrategia(aff,middle, AllVars, BT) :-
```

```
 search(AllVars, 0, anti_first_fail, indomain_middle, complete,
 [backtrack(BT)]).

estrategia(aff,max, AllVars, BT) :-
 search(AllVars, 0, anti_first_fail, indomain_max, complete,
 [backtrack(BT)]).

%-----------------------------------------------------------------------
%estrategia Input order + ordenamiento de valor (min,middle,max)
%-----------------------------------------------------------------------
estrategia(io,min, AllVars, BT) :-
 search(AllVars, 0, input_order, indomain_min, complete,
 [backtrack(BT)]).

estrategia(io,middle, AllVars, BT) :-
 search(AllVars, 0, input_order, indomain_middle, complete,
 [backtrack(BT)]).

estrategia(io,max, AllVars, BT) :-
 search(AllVars, 0, input_order, indomain_max, complete,
 [backtrack(BT)]).

%-----------------------------------------------------------------------
%estrategia Most constrained + ordenamiento de valor (min,middle,max)
%-----------------------------------------------------------------------
estrategia(mc,min, AllVars, BT) :-
 search(AllVars, 0, most_constrained, indomain_min, complete,
 [backtrack(BT)]).

estrategia(mc,middle, AllVars, BT) :-
 search(AllVars, 0, most_constrained, indomain_middle, complete,
 [backtrack(BT)]).

estrategia(mc,max, AllVars, BT) :-
 search(AllVars, 0, most_constrained, indomain_max, complete,
 [backtrack(BT)]).

%-----------------------------------------------------------------------
%contador de backtracks (Round robin)
%-----------------------------------------------------------------------
:- local variable(backtracks), variable(deep_fail).
init_backtracks :-  setval(backtracks,0).
get_backtracks(B) :-getval(backtracks,B).
count_backtracks :- setval(deep_fail,false).
count_backtracks :-
 getval(deep_fail,false),
 setval(deep_fail,true),
 incval(backtracks),
 fail.

%-----------------------------------------------------------------------
%estrategia Round robin + ordenamiento de valor (min,middle,max)
```

```
%-----------------------------------------------------------------------
solve(List,rr,Value_Ord):-
 init_backtracks,
 (foreach(Var,List),param(Value_Ord) do
  count_backtracks,
  indomain(Var,Value_Ord)
 ),
 get_backtracks(B),
 printf("Solucion encontrada con %d backtracks%n", [B]).

%% Inicializar las variables del modelo ---------------------------

initialize(PrevList, CurrList, NextList):-
 flatten_vector(CurrList,RevFlatList),append(PrevList,RevFlatList,
     NextList).

%% Retorna una lista que representa el arreglo o matriz ------------

flatten_vector(CurrList,RevFlatList):-
 term_variables(CurrList, FlatList),rev(FlatList,RevFlatList).

%% Invierte la lista de entrada ------------------------------------

rev(L,R) :- accRev(L,[],R).
accRev([H|T],A,R) :- accRev(T,[H|A],R).
accRev([],A,A).

%-----------------------------------------------------------------------
%Labeling (estrategia por defecto de Eclipse)
%-----------------------------------------------------------------------

estrategia(lb,min, AllVars, BT) :-
 search(AllVars, 0, input_order, indomain, complete, [backtrack(BT)]).
```