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

UNA ARQUITECTURA MODULAR PARA AUTONOMOUS SEARCH

VÍCTOR MANUEL BUSTOS ALLENDES

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

Agosto 2012

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

UNA ARQUITECTURA MODULAR PARA AUTONOMOUS SEARCH

VÍCTOR MANUEL BUSTOS ALLENDES

Profesor Guía: **Ricardo Soto de Giorgis**

Profesor Co-referente: **Broderick Crawford Labrín**

Carrera**: Ingeniería Civil Informática**

Agosto 2012

Dedicatoria

A mis padres, hermanas, familia y amigos que han sido un pilar fundamental durante mi desarrollo personal, espiritual y profesional.

Índice

Resumen

Autonomous Search (AS) dentro de la programación con restricciones, provee la habilidad de reemplazar en forma dinámica estrategias de bajo desempeño por otras más prometedoras. La idea es agilizar los tiempos de resolución en problemas de satisfacción de restricciones. El reemplazo de estrategias es llevado a cabo en base a un ranking de calidad, el cual es calculado por medio de una función de selección. Esta función determina el rendimiento de una estrategia en un tiempo determinado, en base a un conjunto de indicadores y parámetros de control.

En el presente proyecto se propone extender la herramienta actual para AS. Entre otros, se agrega la posibilidad de resolver nuevos problemas, se introduce un generador de gráficos para analizar la traza de la resolución, se modulariza la arquitectura para incluir nuevos optimizadores de parámetros y finalmente se crean los correspondientes manuales de usuario.

Palabras-Claves: *Autonomous Search, Problema de Satisfacción con Restricciones, Estrategias de Enumeración.*

Autonomous Search within constraint programming provides the ability of dynamically replacing low-performance strategies by more promising ones. The idea is to speed-up solving times in the resolution of constraint satisfaction problems. The replacement of strategies is carried out depending on a quality rank, which is calculated by means of a choice function. This function determines the performance of a strategy in a given amount of time via a set of indicators and control parameters.

In the present project we propose to extend the state-of-the-art tool for AS. Among others, we add the possibility of solving new problems, we introduce a chart generator for the analysis of the resolution trace, we modularize the architecture in order to add new parameter optimizers. Finally we include the corresponding user manuals.

Keywords: *Autonomous Search, Constraint Satisfaction Problem, Enumeration Strategies.*

Índice de Ilustraciones

Índice de Tablas

Capítulo 1

Introducción

La programación con restricciones, en inglés Constraint Programming (CP), es un paradigma de programación empleado para el modelado y resolución de problemas complejos, particularmente combinatorios y de optimización. CP es ampliamente utilizado en diferentes áreas de aplicación, tales como en ingeniería para el diseño de estructuras complejas, en gráfica computacional para expresar coherencia geométrica, en sistemas de bases de datos para medir y/o restaurar la consistencia de los datos e incluso para la secuencia de DNA en la biología molecular [1]. Bajo tal paradigma, los problemas se modelan formalmente como problemas de satisfacción de restricciones, en inglés Constraint Satisfaction Problem (CSP). Esta representación consiste principalmente en una secuencia de variables ligadas a un dominio y un conjunto de relaciones sobre tales variables llamadas restricciones.

La resolución de CSPs se lleva a cabo mediante una estructura arbórea que mantiene las potenciales soluciones. Esta estructura se genera en forma dinámica por medio de dos fases: enumeración y propagación. En la fase de enumeración, una variable y un valor son escogidos de su dominio para crear la rama del árbol. Estas decisiones son determinadas por la heurística de selección de variable y valor, respectivamente. En la fase de propagación se aplica un sistema de filtraje para podar el árbol, es decir, los valores que no conducen a ninguna solución son temporalmente eliminados de los dominios. Conjuntamente, la heurística de selección de variable y valor, constituyen lo que se conoce como estrategia de enumeración. La estrategia de enumeración es un elemento clave en el proceso de resolución. Seleccionar una estrategia adecuada puede mejorar notablemente el proceso de búsqueda, sin embargo conocer en forma anticipada el comportamiento de ésta para tomar decisiones correctas es un problema aun abierto en la comunidad.

Autonomous Search (AS) se ha propuesto recientemente para atacar esta problemática. La idea es reemplazar en forma dinámica estrategias de bajo desempeño por otras más prometedoras. Este reemplazo es llevado a cabo en base a un ranking de calidad, el cual es calculado por medio de una función de selección. Esta función determina el rendimiento de una estrategia en un tiempo determinado, en base a un conjunto de indicadores y parámetros de control. El objetivo principal de este proyecto es agregar nuevas funcionalidades a la herramienta actual para AS. Entre otros, se agrega la posibilidad de resolver nuevos problemas, se introduce un generador de gráficos para analizar la traza de la resolución, se modulariza la arquitectura para incluir nuevos optimizadores de parámetros y finalmente se crean los correspondientes manuales de usuario.

El presente trabajo se organiza de la siguiente manera. Se comienza describiendo los objetivos del proyecto, tanto el general como los específicos, los cuales indican lo que comprende la realización del proyecto. Luego se presentará de forma general el concepto de la programación con restricciones, donde se explican algunos ejemplos de CSP y el modelamiento relacionado a éstos, principales algoritmos de búsquedas y estrategias de resolución. Posteriormente se señalan los componentes del AS e información relacionada a la respectiva herramienta. Y finalmente se describen las nuevas funcionalidades que se agregarán a la herramienta actual.

Capítulo 2

Definición de Objetivos

2.1 Objetivo General

Agregar nuevas funcionalidades a la herramienta actual para Autonomous Search dentro del contexto de la programación con restricciones.

2.2 Objetivos Específicos

- Analizar el funcionamiento de la arquitectura actual de Autonomous Search.
- Extender la arquitectura anteriormente propuesta para permitir el ingreso de nuevos problemas.
- Agregar elementos que permitan explotar de mejor manera la nueva arquitectura:
    - ✓ Corrección de bugs.
    - ✓ Generador de gráficos.
    - ✓ Manual de usuario.
    - ✓ Asistente de choice functions.

Capítulo 3

Estado del Arte

La Programación con Restricciones es una de las principales contribuciones de las ciencias de la computación para resolver problemas de alta complejidad. Este paradigma es usado para representar una amplia variedad de problemas que pueden ser modelados como problemas de satisfacción de restricciones y resolverlos a través de la propagación de restricciones y estrategias de enumeración.

La investigación sobre estrategias de enumeración ha sido centro de investigación durante muchos años, principalmente desde su impacto en los procesos de resolución. Desde los años 70 han existido diferentes estudios. Por ejemplo, estudios preliminares se enfocaron en definir un criterio general, es decir, que se utilice el dominio más pequeño para seleccionar la variable, y su valor mínimo, máximo o randómico. Se pueden encontrar trabajos centrados en determinar la mejor estrategia basada en algunos criterios estáticos [2, 4, 11]. Sin embargo, resulta bastante difícil tomar una decisión a priori al igual que predecir su efecto.

Durante los últimos años se ha apreciado una tendencia de analizar el estado de progreso de los procesos de resolución con el fin de que automáticamente se puedan identificar buenas estrategias de desarrollo (o combinaciones de ellas). Por ejemplo, el motor de adaptación con restricción, en inglés adaptative constraint engine (ACE) [9], que es un framework que aprende a ordenar heurísticas mediante la recopilación de la experiencia de la resolución de problemas. La idea principal, es tratar de administrar un conjunto de asesores que recomienden en forma de comentario una acción determinada, por ejemplo, elegir variables con el tamaño de dominio máximo, por lo cual pueden ser vistos como colaboradores en la toma de decisiones.

Otro enfoque interesante, es el grado ponderado de la heurística [10]. La idea es asociar pesos de restricciones, los cuales son incrementados durante la propagación de la arco-consistencia (ver capítulo 5). La suma de pesos es calculado por cada variable involucrada en las restricciones y la variable con la suma más grande es seleccionada. El principio que soporta el procedimiento de medida ponderada puede ser concebido en términos de una estrategia general que combina dos heurísticas principales: el principio fail-first y el principio de contención. El principio de fail-first dice: para tener éxito, se debería buscar primero donde es más probable fallar [12] mientras que el principio de contención dice: es más probable causar el fallo si se eligen las variables que están directamente relacionadas con el fracaso.

Autonomous Search opera bajo un framework que desarrolla una propia adaptación del proceso de búsqueda cuando éste exhibe un pobre rendimiento. Esto consiste básicamente en analizar de manera eficiente el estado del proceso de resolución de un

determinado problema. El reemplazo, es desarrollado en base a un ranking de calidad, el cual es calculado por medio de la función de selección. Ésta determina el rendimiento de una estrategia en un tiempo determinado, en base a un conjunto de indicadores y parámetros de control. El objetivo principal de este proyecto es agregar nuevas funcionalidades a la herramienta actual para AS. Entre otros, se agrega la posibilidad de resolver nuevos problemas, se introduce un generador de gráficos para analizar la traza de la resolución, se modulariza la arquitectura para incluir nuevos optimizadores de parámetros y finalmente se crean los correspondientes manuales de usuario.

Capítulo 4

Problema de Satisfacción de Restricciones

4.1 Definición de un Problema de Satisfacción de Restricciones

Un problema de satisfacción de restricciones puede ser representado mediante una terna (X, D, C) donde [1]:

X es un conjunto de n variables $\{X_1, \dots, X_n\}$.

$D =< D_1, \dots, D_n >$ es una tupla de dominios finitos donde se interpretan las variables X, tal que la i-ésima componente D_i es el dominio que contiene los posibles valores que pueden asignarse a la variable x_i. La cardinalidad de cada dominio es $d_i = |D_i|$.

$C = \{c_1, c_2, \dots, c_p\}$ es un conjunto finito de restricciones. Cada restricción *k*-aria c_i está definida sobre un conjunto de *k* variables $var(c_i) \subseteq X$, denominando su ámbito, y restringe los valores que dichas variables pueden simultáneamente tomar. Particularmente una restricción es binaria cuando relaciona únicamente a dos variables x_i y y_i, y se puede denotar como c_{ij}. Todas las restricciones definidas en un CSP son conjuntivas, de manera que una solución debe satisfacer a todas ellas.

La *instanciación* de una variable es un par variable-valor (x, a) que representa la asignación del valor a a la variable x $(x = a)$. La instanciación de un conjunto de variables es una tupla de pares ordenados, donde cada par ordenado (x_i, a_i) asigna el valor $\{a_i \in D_i\}$ a la variable x_i . Una tupla $\big((x_1, a_1), \dots, (x_i, a_i)\big)$ es localmente consistente si satisface todas las restricciones formadas por variables $\{x_1, \dots, x_i\}$ de la tupla. Para simplificar la notación, se sustituirá la tupla $\big((x_1, a_1), \dots, (x_i, a_i)\big)$ por $(a_1, \dots, a_i)$.

Un valor $a \in D_i$ es un valor consistente para x_i si existe al menos una solución del CSP en la cual $x_i = a_i$. El dominio mínimo de una variable x_i es el conjunto de todos los valores consistentes para la variable, es decir, quedan excluidos aquellos valores que no forman parte de ninguna solución. La siguiente notación $(\forall\, x_i \in X, \forall\, a \in D_i, x_i = a\,)$ forma parte de una solución del CSP

En la ilustración 1 [14], se presenta la forma general de resolver un problema de satisfacción de restricciones. También se señala la propagación con restricciones, cuyo objetivo es inferir nueva información sobre el problema a partir de la que ya se ha explicitado. La forma elemental de este tipo de inferencia consiste en eliminar aquellos valores de los dominios de las variables que no pueden participar en una solución.

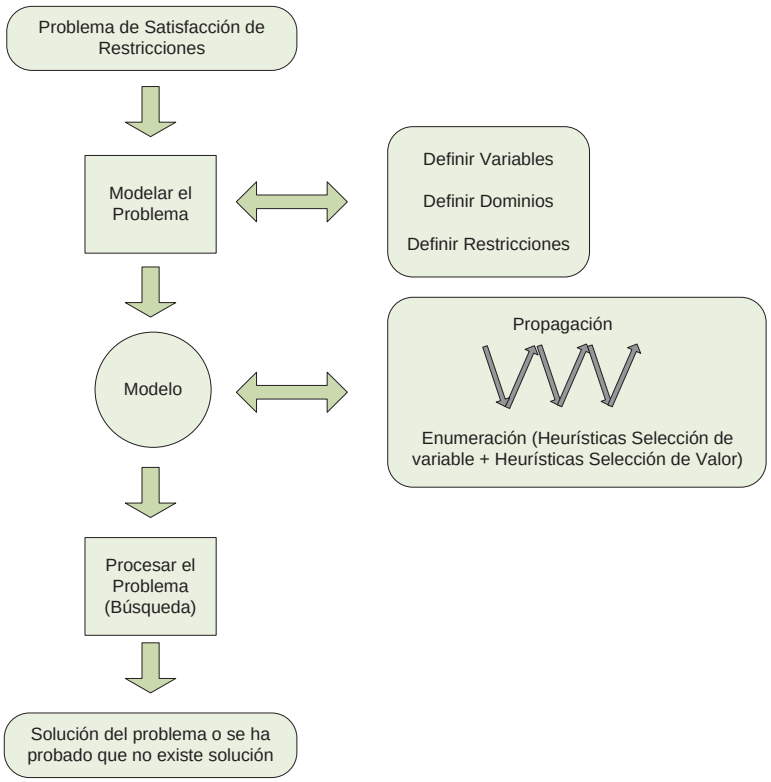


Ilustración 1: Esquema de representación para resolver en forma general un problema de satisfacción de restricciones.

Una solución a un CSP es una asignación $a_1, a_2, \dots, a_n$ de valores a todas sus variables, de tal manera que se satisfagan todas las restricciones del CSP. Es decir, una solución es una tupla consistente que contiene todas las variables del problema. Una solución parcial es una tupla consistente que contiene algunas de las variables del problema. Un CSP es *consistente*, si tiene al menos una solución, es decir una tupla consistente. Dos CSPs son *equivalentes* si ambos representan el mismo conjunto de soluciones.

Ejemplo 4.1.1 Para el problema de las 4-reinas, existen solamente dos soluciones, que se detallan en la ilustración 2. Asumiendo que las variables $\{x_1, x_2, x_3, x_4\}$ representan las columnas, y sus dominios son las posibles filas $\{1,4\}$ donde colocar las reinas, tenemos que: $(x_1, 3)$ es una instanciación de la variable x_1 y $\big((x_1, 2), (x_2, 4), (x_3, 1), (x_4, 3)\big)$ es una solución del CSP, por lo que el CSP es consistente. El valor 3 es un valor consistente para x_1, pero el valor 1 no lo es. El dominio mínimo de x_2 es $\{1,4\}$.

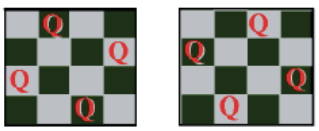


Ilustración 2: Dos soluciones al problema de las cuatro reinas.

Los objetivos del CSP consisten en la *satisfabilidad* del mismo, es decir, obtener una o varias soluciones, sin preferencia alguna, o bien obtener una solución óptima, o al menos una buena solución, en base a una función objetivo previamente definida en términos de algunas o todas las variables.

4.2 Definición y Tipología de las Restricciones

Las restricciones se caracterizan fundamentalmente por su *ariedad* [1], que es el número de variables involucradas en dicha restricción. Una restricción unaria es una restricción sobre una sola variable. Una restricción binaria es una restricción que consta de dos variables. Una restricción no binaria (o n-aria) es una restricción que involucra a un número arbitrario de tres o más variables.

Ejemplo 4.2.1 La restricción $x \leq 5$ es una restricción unaria sobre la variable x. La restricción $x_4 - x_3 \neq 3$ es una restricción binaria. La restricción $2x_1 - x_2 + 4x_3 \geq 4$ es una restricción ternaria.

Una restricción sobre un conjunto de variables puede definirse *extensionalmente* mediante un conjunto de tuplas válidas o no válidas y también *intensionalmente* mediante una función aritmética. La representación extensional es una restricción *k*-aria que está formada por un conjunto de tuplas, cada una con *k* elementos, y expresa el conjunto de valores que las *k* variables pueden tomar simultáneamente. Claramente, en el caso de CSP continuos es imposible representar las restricciones extensionalmente ya que generalmente hay un número infinito de tuplas válidas.

Ejemplo 4.2.2 Considerar una restricción que involucra a las variables x_i, x_2, x_3, x_4 con dominios {1,2}, donde la suma entre las variables x_1 y x_2 es menor o igual que la suma entre x_3 y x_4. Esta restricción puede representarse intensionalmente mediante la expresión $x_1 + x_2 \leq x_3 + x_4$. También, podría representarse extensionalmente mediante el conjunto de tuplas permitidas { (1,1,1,1), (1,1,1,2), (1,1,2,1), (1,1,2,2), (2,1,2,2), (1,2,2,2), (1,2,1,2), (1,2,2,1), (2,1,1,2), (2,1,2,1), (2,2,2,2) }.

En el caso de una definición intensional, se puede tener restricciones *no-disyuntivas* o *disyuntivas*, según expresen una única relación o más de una relación disyuntiva entre las variables. Por ejemplo, asumiendo un modelo con las relaciones elementales $\{<, =, >\}$, $(x_1 < x_2)$, es una restricción no-disyuntiva, mientras que $(x_1 < x_2 \vee x_1 > x_2)$, es decir, $x_1 \neq x_2$ sería una restricción disyuntiva. Por otra parte, las restricciones también pueden ser *cualitativas*, cuando expresan una relación de orden entre las variables (por ejemplo, $x_1 < x_2 + 7$). Las restricciones métricas requieren de una métrica en el dominio de interpretación de las variables.

Un tipo especial de restricciones son las restricciones lineales. Una relación lineal sobre $X = \{x_1, \dots, x_k\}$ es una expresión de la forma:

$$\sum_{i=1}^{k} p_i \, x_i \{<, \leq, \neq, \geq, >\} b$$

Donde p_i son los coeficientes y $b \in \mathbb{R}$. En base a combinaciones lógicas de desigualdad $(\neq)$ e inigualdad $(\leq)$, se pueden expresar todas las relaciones en $\{<, \leq, \neq, \geq, >\}$. Las *Restricciones Lineales Disyuntivas* (DLR) son disyunciones de restricciones lineales.

4.3 Ejemplos de CSP y su Modelamiento

El primer paso en la resolución de un CSP es su modelamiento, es decir, su representación en términos de variables, dominios y restricciones. Al igual que ocurre con el lenguaje natural, el modelamiento de un problema se puede realizar de muchas maneras diferentes. Respecto al modelamiento de un problema CSP hay dos aspectos básicos [1]:

- La potencia expresiva de las restricciones, es decir, la capacidad de modelar las restricciones realmente existentes en el problema real.

- La eficiencia de la representación, ya que dependiendo del modelamiento CSP, el problema se resolverá con más o menos eficiencia.

4.3.1 Coloración del mapa

Este problema parte de un conjunto de colores posibles para colorear cada región del mapa, de manera que regiones adyacentes tengan distintos colores. En la formulación del CSP, se define una variable por cada región del mapa, y el dominio de cada variable es el conjunto de colores disponibles. Para cada par de regiones contiguas existe una restricción sobre las variables correspondientes que no permite la asignación de idénticos valores a las variables. En el caso de la ilustración 3 (lado izquierdo), se tiene un mapa con cuatro regiones x, y, z, w para ser coloreadas con los posibles colores Rojo (r), Verde (v), Azul (a). La formulación CSP sería:

- Variables $\{x, y, z, w\}$
- Dominio $\{Rojo, Verde, Azul\}$, único para las tres variables.
- Restricciones (definición intensional): $\{x \neq y, x \neq w, x \neq z, y \neq w, w \neq z\}$

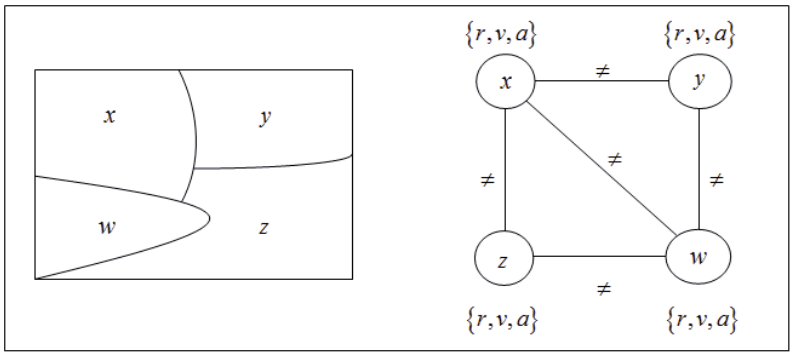


Ilustración 3: El problema de coloración del mapa.

Un CSP binario, puede ser representado mediante una red de restricciones, donde los nodos representan las variables y los arcos representan las restricciones entre las mismas. En la Ilustración 3 (lado derecho), se representa la red correspondiente al problema del ejemplo, donde las variables correspondientes a regiones adyacentes que están conectadas por una arista. Hay cinco restricciones en el problema, es decir, cinco aristas en la red. Una solución para el problema es la asignación $(x, r); (y, v); (z, v); (w, a)$. En esta asignación, todas las variables adyacentes tienen valores diferentes.

4.4.2 Criptografía

El típico problema criptográfico *"send+more=money"* consiste en asignar a cada letra $\{s, e, n, d, m, o, r, y\}$ un dígito diferente del conjunto $\{0, \dots, 9\}$ de forma que se satisfaga: *send + more= money*.

$$\begin{array}{cccccc} & \mathrm{s} & e & n & d \\ + & m & o & r & e \\ \hline m & o & n & e & y \end{array}$$

Las manera más fácil de modelar este problema es asignando una variable a cada una de las letras, todas ellas con un dominio $\{0, \dots, 9\}$ y con las restricciones que obliguen a que todas las variables tomes valores distintos y con la correspondiente restricción para que se satisfaga *"send+more=money"*. De esta forma las restricciones son:

- $10^3(\mathrm{s} + \mathrm{m}) + 10^2(\mathrm{e} + \mathrm{o}) + 10(\mathrm{n} + \mathrm{r}) + \mathrm{d} + \mathrm{e} = 10^4\mathrm{m} + 10^3\mathrm{o} + 10^2\mathrm{n} + 10^3\mathrm{e} + \mathrm{y}$;
- Restricción de todas diferentes: $\neq (s, e, n, d, m, o, r, y)$;

La restricción de "todas diferentes" puede reemplazarse por un conjunto de restricciones binarias, $\{s \neq e, s \neq n, \dots, r \neq y\}$ donde $\left(x_i \neq x_j\right)$ es la relación disyuntiva $x_i\{<, >\}x_j$.

Sin embargo, para el algoritmo de resolución más general como es Backtracking (que se verá en la siguiente sección), este modelo no es muy eficiente porque todas las variables necesitan ser instanciadas antes de comprobar estas dos restricciones. De esta manera no se podría podar el espacio de búsqueda durante el propio proceso a fin de agilizar la búsqueda. Además, la primera restricción es una igualdad en la que forman parte todas las variables del problema (restricción global) por lo que dificulta el proceso de consistencia. Un modelo más eficiente podría utilizar los bits de acarreo para descomponer la ecuación anterior en un conjunto de pequeñas restricciones. Esto requeriría incluir tres variables *portadoras* adicionales c_1, c_2, c_3 cuyo dominio es $\{0,1\}$:

- $e + d = y + 10c_1$;
- $c_1 + n + r = e + 10c_2$;
- $c_2 + e + o = n + 10c_3$;
- $c_3 + s + m = 10m + o$;
- Restricción de todas diferentes: $\neq (s, e, n, d, m, o, r, y)$;

Capítulo 5

Algoritmos de Búsqueda

Los algoritmos de búsqueda se encargan de realizar la búsqueda exhaustiva a través del espacio de las posibles asignaciones en busca de aquellas que cumplan de forma simultánea con todas las restricciones del problema, estos algoritmos realizan una búsqueda completa garantizando encontrar una solución y en caso contrario demostrando la no existencia de las mismas. El esquema de funcionamiento general de un algoritmo de búsqueda se muestra a continuación:

- Explora el espacio de estados del problema (Configuraciones posibles)
- Termina cuando:
    - Encuentra una solución
    - Demuestra que no hay solución
    - Agota los recursos computacionales

En la literatura se han desarrollado variadas alternativas de algoritmos de búsqueda, algunos ejemplos orientados a CSPs binarios como: 'Backtracking Cronológico', 'Back Jumping', 'Conflict-Directed Backtraking', 'Backtracking Dinámico', 'Forward Checking', 'Minimal Forward Checking', 'Forward Checking con Conflict Directed Backtracking' y 'Manteniendo Arco-Consistencia'(MAC) [1].

Algunos de los algoritmos anteriores se han extendido a CSP no binarios. Por ejemplo hay varias extensiones de Forward Checking para problemas no binarios. Además MAC se ha extendido a un algoritmo que mantiene la arco-consistencia [1]generalizada sobre restricciones de cualquier aridad. En la siguiente sección se describen los algoritmos de búsqueda más utilizados:

5.1 Árbol de Búsqueda

Las posibles combinaciones de la asignación de valores a las variables en un CSP generan un espacio de búsqueda que puede ser visto como un árbol de búsqueda [1]. La búsqueda mediante backtracking en un CSP corresponde a la tradicional exploración primero en profundidad en el árbol de búsqueda. Si se asume que el orden de las variables es estático y no cambia durante la búsqueda entonces el nodo en el nivel k del árbol de

[1] Un problema binario es arco-consistente si para cualquier par de variables restringidas x_i y x_j, para cada valor a en D_i hay al menos un valor b en D_j tal que las asignaciones (x_i, a) y (x_j, b) satisfacen la restricción entre x_i y x_j [13].

búsqueda representa un estado donde las variables $x_1, \dots, x_k$ están asignadas y el resto $x_{k+1}, \dots, x_n$ no lo están. Se puede asignar cada nodo en el árbol de búsqueda con la tupla consistente de todas las asignaciones llevadas a cabo. La raíz del árbol de búsqueda representa la tupla vacía (ejemplo (-,-,-), para 3 variables), donde ninguna variable tiene asignado valor alguno.

Los nodos en el primer nivel son 1-tuplas que representan estados donde se les ha asignado un valor a la variable x_1 (ejemplo (0,-,-), para 3 variables), los nodos en el segundo nivel son 2-tuplas que representan estados donde se han asignado valores a las variables x_1 y x_2 (ejemplo (0,0,-), para 3 variables), y así sucesivamente. Si n es el número de variables del problema, los nodos en el nivel n, que representan las hojas del árbol de búsqueda son n-tuplas, que representan la asignación de valores para todas las variables del problema. De esta manera, si una n-tupla es consistente, entonces es solución del problema. Un nodo del árbol de búsqueda es consistente si la asignación parcial actual es consistente, o en otras palabras, si la tupla correspondiente a ese nodo es consistente. Los nodos que se encuentran próximos a la raíz, se les llama nodos superficiales. Los nodos próximos a las hojas del árbol de búsqueda se les llaman nodos profundos.

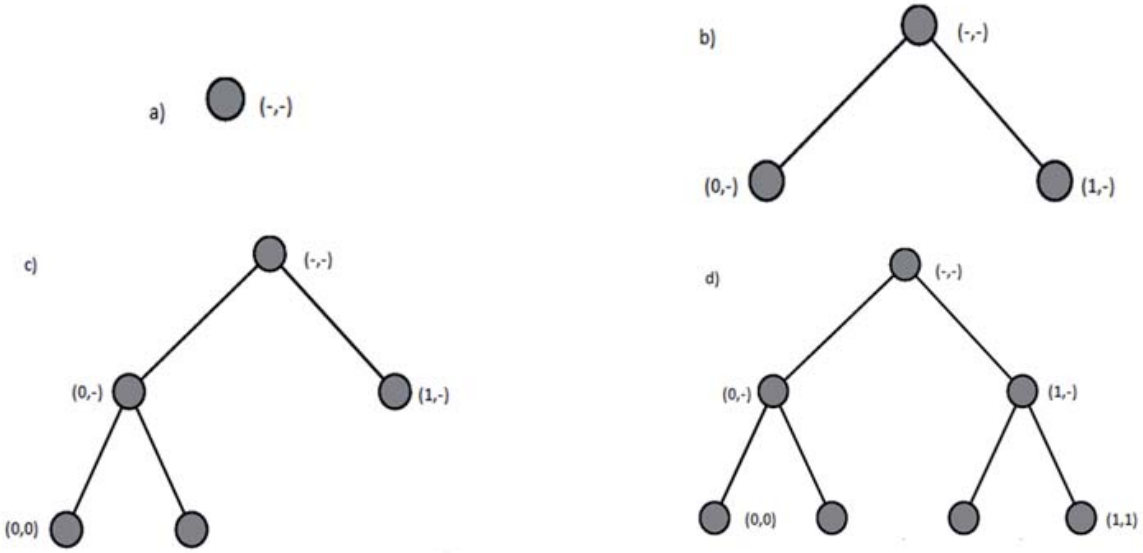


Ilustración 4: Secuenciación de árbol de búsqueda.

5.2 Métodos de Búsqueda

Los métodos de búsqueda se centran en explorar el espacio de estados del problema. Estos métodos pueden ser *completos*, explorando todo el espacio de estados en busca de una solución, o *incompletos* si solamente exploran una parte del espacio de estados. Los métodos que exploran todo el espacio de búsqueda garantizan encontrar una solución, si existe, o demuestran que el problema no es resoluble [15]. La desventaja de estos algoritmos es que son muy costosos [13].

5.2.1 Generar y Testear (GT)

Este método genera las posibles tuplas de instanciación de todas las variables de forma sistemática y después testea sucesivamente sobre cada instanciación si se satisfacen todas las restricciones del problema. La primera combinación que satisfaga todas las restricciones, será la solución al problema [1]. La desventaja de este algoritmo es que realiza muchas instanciaciones erróneas [2] de valores a variables que después son rechazadas en la fase de testeo.

5.2.2 Backtracking Cronológico (BT)

El algoritmo BT mejora el algoritmo anterior de la siguiente forma: cada vez que se asigna un nuevo valor a la variable actual (x_i), se comprueba si es consistente con los valores que han sido asignados a las variables pasadas. Si no lo es, se abandona esta asignación parcial, y se asigna un nuevo valor a x_i, si ya se han agotado todos los valores de D_i , BT retrocede para probar otro valor para la variable x_{i-1}, si se ha agotado D_{i-1}, retrocede al nivel $i-2$, y así sucesivamente hasta encontrar una asignación de un valor a una variable que es consistente con las variables pasadas o hasta que se demuestra que no hay más soluciones [5].

Es decir, BT recorre el árbol utilizando búsqueda primero en profundidad y, en cada nodo, comprueba si la variable actual es consistente con las variables pasadas. Si detecta inconsistencia descarta la asignación parcial actual puesto que no es parte de ninguna asignación completa que sea solución. De esta forma, se ahorra recorrer el subárbol que cuelga de esta asignación parcial [7].

Problemas con el algoritmo de Backtracking Cronológico:

- Tiene una visión local del problema solo comprueba restricciones que están formadas por la variable actual y las pasadas, e ignora la relación entre la variable actual y las futuras.
- Este algoritmo no recuerda movimientos previos, y por lo tanto los podría volver a repetir.
- Resulta ser un algoritmo muy ineficiente.

Para ayudar a combatir estos inconvenientes, se han desarrollado algunos algoritmos de búsqueda más robustos. Estos algoritmos se pueden dividir en algoritmos look-back y look-ahead. La Ilustración 5 [1] muestra un ejemplo de BT.

5.3 Algoritmos Look-Backward

Al igual que Backtracking Cronológico, los algoritmos Look-Backward realizan la comprobación de consistencia hacia atrás, es decir, entre la variable actualmente instanciada y las pasadas ya instanciadas. Pero la diferencia es que Look-Backward trata de explotar la información del problema para comportarse de mejor manera frente a

situaciones sin salida. Algunas variantes del algoritmo Look-Backward son: Backjumping, Conflict-Directed, Learning [1].

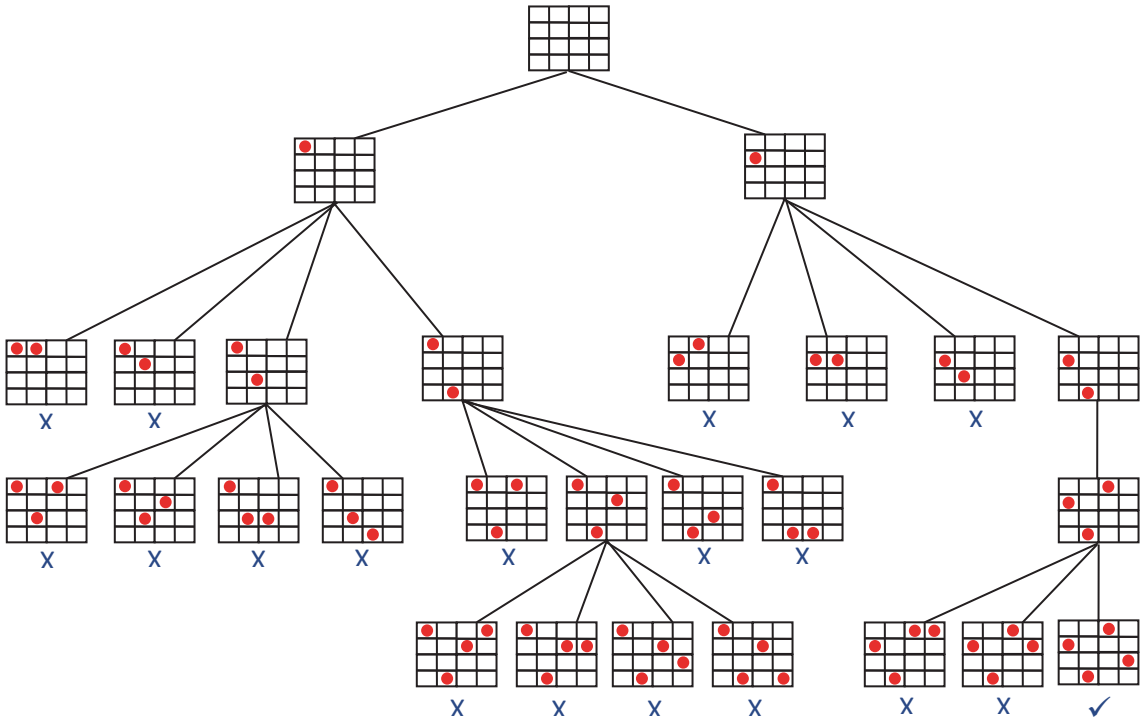


Ilustración 5: Backtracking Cronológico aplicado al problema de las 4-Reinas.

5.3.1 Backjumping (BJ)

Es un algoritmo para CSPs parecido al backtracking cronológico excepto que se comporta de una manera más inteligente cuando encuentra situaciones sin salida. En vez de retroceder a la variable anteriormente instanciada, BJ salta a la variable más profunda (más cerca de la variable actual) x_j que está en conflicto con la variable actual x_i donde $j < i$. Se dice que una variable instanciada x_j está en conflicto con una variable x_i si la instanciación de x_j evita uno de los valores en x_i (debido a la restricción entre x_i y x_j). Cambiar la instanciación de x_j puede hacer posible encontrar una instanciación consistente de la variable actual [1].

5.3.2 Conflict-directed Backjumping (CBJ)

Tiene un comportamiento de salto hacia atrás más sofisticado que BJ. Cada variable x_i tiene un conjunto conflicto formado por las variables pasadas que están en conflicto con x_i. En el momento en el que la comprobación de la consistencia entre la variable actual x_i. En una situación sin salida, CBJ salta a la variable más profunda en su conjunto, por ejemplo x_k donde $k < i$. Al mismo tiempo se incluye el conjunto conflicto de x_i al de x_k, por lo que no se pierde ninguna información sobre los conflictos. Obviamente, CBJ

necesita unas estructuras de datos más complicadas para almacenar los conjuntos conflicto [1].

5.3.3 Learning

Es un método que almacena las restricciones implícitas que son derivadas durante la búsqueda y las usa para podar el espacio de búsqueda. Por ejemplo, cuando se alcanza una situación sin salida en la variable x_i entonces se sabe que la tupla de asignaciones $\big((x_1, a_1), \dots, (x_{i-1}, a_{i-1})\big)$ lleva a una situación sin salida. Así, se puede aprender que una combinación de asignaciones para las variables $x_1, \dots, x_{i-1}$ no está permitida [1].

5.4 Algoritmos Look-Ahead

Como ya se señaló anteriormente, los algoritmos Look-Backward tratan de reforzar el comportamiento de BT mediante un comportamiento más inteligente cuando se encuentran en situaciones sin salida. Sin embargo, todos ellos todavía llevan a cabo la comprobación de la consistencia solamente hacia atrás, ignorando las futuras variables. Los algoritmos Look-Ahead hacen una comprobación hacia adelante en cada etapa de la búsqueda, es decir, llevan a cabo las comprobaciones para obtener las inconsistencias de las variables futuras involucradas además de las variables actual y pasadas. De esa manera, las situaciones sin salida se pueden identificar antes y además los valores inconsistentes se pueden descubrir y podar para las variables futuras [1].

5.4.1 Forward Checking (FC)

Es uno de los algoritmos Look-Ahead más comunes. Es un caso particular de verificación de arco-consistencia. Un estado es arco-consistente si el valor de cada variable es consistente con las restricciones sobre la variable en particular. La arco-consistencia es obtenida por sucesivas eliminaciones de valores inconsistentes.

En cada etapa de la búsqueda BT, FC comprueba hacia adelante la asignación actual con todos los valores de las futuras variables que están restringidas con la variable actual. Los valores de las futuras variables que son inconsistentes con la asignación actual son temporalmente eliminados de sus dominios. Si el dominio de una variable futura se quedo vació, la instanciación de la variable actual se deshace y se prueba con un nuevo valor. Si ningún valor es consistente, entonces se lleva a cabo el Backtracking Cronológico [1]. FC garantiza que en cada etapa, la solución parcial actual es consistente con cada valor de cada variable futura. Además, cuando se asigna un valor a una variable, solamente se comprueba hacia adelante con las futuras variables con las que están involucradas. Así mediante la comprobación hacia adelante, FC puede identificar antes las situaciones sin salida y podar el espacio de búsqueda. El proceso de Forward Checking se puede ver cómo aplicar un simple paso de verificación de arco-consistencia sobre cada restricción que involucra la variable actual con una variable futura después de cada asignación de variable. La Ilustración 6 [19] presenta el pseudocódigo del algoritmo Forward Checking.

Procedimiento 1 *FC* (*k* :integer, *inst* : array)

```
1:  while D[k] ≠ { } and not success do
2:      a ← choose_from( D[k] )
3:      inst ← instantiate( inst , k , a )
4:      if consistent( inst , k , success) then
5:         if success then
6:             print solution( inst )
7:         Else
8:             propagate( k , D , failure)
9:         end if
10:        if not failure then
11:           FC ( k +1, inst )
12:        end if
13:     end if
14:     restore( k )
15: end while
```

Ilustración 6: Pseudocódigo del algoritmo Forward Checking.

Forward Checking se ha combinado con algoritmos look-back para generar nuevos algoritmos. Por ejemplo, Forward Checking con Conflict Directed Backjumping (FC-CBJ) es un algoritmo que combina el movimiento hacia adelante de FC con el movimiento hacia atrás de CBJ, y de esa manera tiene las ventajas de ambos algoritmos

5.4.2 Minimal Forward Checking (MFC)

Es una versión de FC que retrasa llevar a cabo toda la comprobación de la consistencia de FC hasta que es absolutamente necesario. En vez de comprobar hacia adelante la asignación actual contra todas las variables futuras, MFC sólo comprueba si la asignación actual causa una limpieza de dominios. Para hacer esto es suficiente comprobar la asignación actual con los valores de cada variable futura hasta que se encuentra una que es consistente. Después, si el algoritmo ha retrocedido, vuelve atrás y lleva a cabo las comprobaciones 'perdidas'. Claramente, MFC siempre lleva a cabo a lo sumo el mismo número de comprobaciones que FC. Resultados experimentales han demostrado que la ganancia no supera 10% [1].

5.4.3 Maintaining Arc Consistency (MAC)

El procedimiento realizado en el algoritmo FC puede ser sin duda más ambicioso, por ejemplo, si no verifica solamente los conflictos entre la actual y las futuras variables, pero también entre los conflictos entre las futuras variables. Este enfoque es propuesto en el procedimiento Maintaining Arc Consistency, también conocido como Full Look Ahead [19].

El proceso MAC es descrito en la Ilustración 7 [19]. Este claramente poda el espacio de búsqueda antes que FC, pero muy superior a la labor realizada en cada instanciación. Por ejemplo, colocando la primera reina en la posición (1,1) es suficiente

para determinar un error. Tal instanciación elimina los conflictos de celda de la primera fila y diagonal. Después de eso, los conflictos entre las variables futuras son verificados.

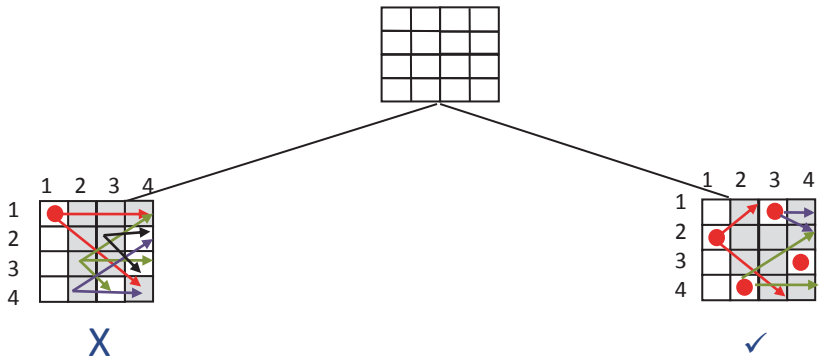


Ilustración 7: MAC aplicado al problema de las 4-Reinas.

El procedimiento comienza con la primera posición disponible en la segunda columna, el cual corresponde a la celda $(3,2)$. Esto se establece como inaceptable, puesto que no deja lugar para la tercera reina. El algoritmo sigue con la celda $(4,2)$, la próxima posición disponible en la segunda columna. Esta colocación permite que la celda $(2,3)$ sea la única posición abierta en la tercera columna, que se establece luego como incompatible, ya que no es posible localizar la cuarta reina. El proceso sigue hasta que el resultado se alcanza en el subárbol derecho.

Capítulo 6

Estrategias de Enumeración

Lo más habitual para la resolución de un CSP consiste en alternar fases de propagación y enumeración. En esta última se debe establecer el orden en el cual se van a considerar las variables, así como el orden en el cual se van a instanciar los valores de los dominios de cada una de las variables. Para establecer dichos órdenes, se utilizan heurísticas de selección de variable y heurísticas de selección de valor respectivamente, las que en conjunto constituyen las denominadas *Estrategias de Enumeración* [1]. En la literatura se ha establecido que seleccionar un orden correcto de las variables y de los valores, puede mejorar considerablemente la eficiencia de resolución. Es por esto que existen diversos esfuerzos en definir este tipo de heurísticas, algunas de las cuales se describen en la presente sección.

6.1 Heurísticas de Selección de Variable

La idea principal que existe tras la elección de una variable, es minimizar el tamaño del árbol de búsqueda (espacio de búsqueda mejorado) y asegurar que aquellas ramas del árbol que no tienen solución, serán podadas tan pronto como sea posible. Esto es denominado por Haralick y Elliot, como el principio Fail-First, el cual se describe como: *Para tener éxito, intente primero donde es más probable fallar*.

La selección de variables puede ser dinámica o estática, donde los términos dinámicos y estáticos hacen referencia al momento en el que se establece el orden en que se considerarán las variables para instanciación, definición que coincide con la utilizada en [1], y la cual difiere del concepto utilizado en [6], donde la idea de dinamismo está basada en la satisfacción de restricciones adaptativas.

6.1.1 Heurísticas de Selección Estáticas

Estos tipos de heurísticas generan un orden fijo de las variables antes de iniciar la búsqueda, y las variables son siempre seleccionadas en el orden predefinido para instanciación. Sólo explotan la información del estado inicial de la búsqueda.

- **Minimun Width (MW) [1]:** La selección de variables se realiza según la ordenación lineal de menor anchura del grafo de restricciones. Mediante este orden, se persigue que cuando se asigne una variable, sus padres estén ya asignados, de forma que las situaciones sin salida puedan identificarse antes y se reduzca el número de vueltas atrás.

- **Maximun Degree (MD) [1,21]:** Conocida también como Max – Static –Degree. Esta heurística selecciona las variables decrecientemente según su grado con el

grafo de restricciones original. Se entiende como grado de una variable al número de variables con las que está conectada. De esta forma, esta heurística selecciona primero las variables de mayor grado, es decir, las más conectadas.

- **Minimum Domain Variable (MDV) [1]:** Esta heurística selecciona las variables de acuerdo a la menor cardinalidad de su dominio. Las variables con dominio más pequeño son elegidas antes. Sin embargo, respecto a la aplicación de esta heurística, es preferible su versión dinámica (*minimun remaining values*).

- **Min Domain / Degree [21]:** Elige la variable que minimiza la proporción entre el tamaño del dominio y el grado de la variable.

6.1.2 Heurísticas de Selección Dinámicas

El problema de los algoritmos de ordenación de variables estáticos es que no tienen en cuenta los cambios en los dominios o relaciones de las variables causados por la propagación de las restricciones durante la búsqueda. Por ello, estas heurísticas generalmente se utilizan en algoritmo de comprobación hacia atrás (o Look-Backward) donde no se lleva a cabo la propagación de las instanciaciones que se van realizando.

Estas heurísticas pueden cambiar el orden de instanciación de las variables dinámicamente a medida que se avanza en el árbol de búsqueda. Para generar dicho orden, se basa en información generada durante la búsqueda.

- **Minimun Remaining Values (MRV) [1]:** En cada paso, se selecciona la variable con los dominios de instanciación más pequeños. Esta heurística se basa en la intuición de que si una variable tiene pocos valores de elección en su dominio, entonces es más difícil encontrar un valor consistente.

- **Maximun Cardinality (MC) [1]:** Esta heurística, conocida como Max Backward Degree, consiste en seleccionar la primera variable arbitrariamente y luego selecciona la variable que está relacionada con el mayor número de variables ya instanciadas. La intuición es que resulta la más restringida, al estar relacionada con el mayor número de variables ya instanciadas.

- **Maximun Forward Degree [20,21]:** La variable seleccionada es aquella que maximiza el número de variables adyacentes no instanciadas.

- **Domdeg [20]:** Esta heurística es equivalente a Min Domain / Forward Degree, esto significa que se selecciona la variable que minimiza la proporción entre el tamaño de dominio y el forward degree, este último correspondiente al número de variables adyacentes no instanciadas.

6.2 Heurísticas de Selección de Valor

En la elección de valor, se puede elegir, un valor que sea más probable de llevar a una solución y así reducir el riesgo de tener que hacer backtracking y probar un valor alternativo. En la práctica, por supuesto, lo mejor que se puede hacer es elegir un valor que es menos probable que nos conduzca a una falla. Este principio, llamado *succeed – first*, no tiene una heurística de selección de valor ampliamente aplicable comparado a la heurística MRV, pertinente a *fail – first*, pero puede dar buenas heurísticas aplicables a problemas individuales, o tipos de problemas. En síntesis, este principio indica que el valor con alto número de soportes es preferible.

- **Min – Conflicts [5,22]:** Como su nombre lo indica, se selecciona el valor que genera los mínimos conflictos para asignaciones futuras. En otras palabras, se cuenta la cantidad de valores del dominio de cada variable adyacente, que son incompatibles con el valor seleccionado, se elige aquel que suma la cantidad más baja. Si hay más de un mínimo, entre ellos se elige uno aleatoriamente.

- **Max Domain Size [3,5]:** Selecciona el valor que deja los máximos tamaños de dominio para las variables futuras.

- **Weighted Max Domain Size [3,5]:** Se especifica la manera en que se eligen, en caso de que existan empates al usar Max Domain Size, los valores. Por ello se considera el tamaño de los dominios. Por ejemplo, si al tomar un valor a_j se deja a cuatro futuras variables con tamaño de dominio igual a cuatro, se selecciona el valor a_j.

- **Point Domain Size [3,5]:** A cada valor de la variable actual se le asigna un peso, este depende del número de futuras variables que quedaran con cierto tamaño de dominio. Este peso es conocido como punto de valor. Por ejemplo, se toma un valor y por cada dominio que quede de tamaño uno, se le asignan ocho puntos de valor a dicha variable. Por cada dominio que genere de tamaño dos, se le asignan cuatro puntos de valor a dicha variable. Por cada dominio que genere de tamaño cuatro, se le asigna un punto de valor a la variable. Se aprueba con todos los valores para la variable actual. Finalmente, se elige el valor con la menor cantidad de puntos de valor, o peso.

- **Promise [5]:** Para cada valor a_j de la variable x_i se cuenta el número de valores que hay compatibles con a_j en cada variable adyacente futura, y se toma el producto de las cantidades contadas. Este producto se conoce como la promesa de valor. La heurística seleccionará el valor con la máxima promesa.

En las tablas expuestas a continuación, se muestra la justificación de las heurísticas de selección de variables [5] y valor [5] anteriormente descritas, estas justificaciones de las heurísticas se han basado en los principios teóricos que las sustentan, ya sea *fail-first* para selección de variable o *succeed – first* para selección de valor.

Heurística de Selección de Variable	Justificación
Minimum Widht	Reducir el número de Backtracking (Fail - First)
Maximum Degree	Reducir el número de Backtracking (Fail - First)
Minimum Domain Variable	Encontrar incosistencias lo antes posible (Fail - First)
Min Domain / Degree	Encontrar incosistencias lo antes posible (Fail - First)
Minimum Reamining Values	Encontrar incosistencias lo antes posible (Fail - First)
Maximun Forward Degree	Encontrar inconsistencias lo antes posible (Fail - First)
Domdeg	Encontrar inconsistencias lo antes posible (Fail - First)

Tabla 1: Justificación para heurística de selección de variable.

Heurística de Selección de Valor	Justificación
Min - Conflicts	Seleccionar valor menos restringido (succeed - first)
Max Domain Size	Dejar tamaños de dominios máximos (succeed - first)
Weighted Max Domain Size	Dejar tamaños de dominios máximos (succeed - first)
Point Domain Size	Dejar tamaños de dominios máximos (succeed - first)
Promise	Seleccionar valor menos restringido (succeed - first)

Tabla 2: Justificación para heurística de selección de valor.

Capítulo 7

Solvers

Los lenguajes de programación con restricciones son típicamente ampliaciones de otro lenguaje. El primer lenguaje utilizado para tal efecto fue Prolog. Por esta razón es que este campo fue llamado inicialmente Programación Lógica con Restricciones. Ambos paradigmas comparten características muy similares, tales como las variables lógicas (una vez que una variable es asignada a un valor, no puede ser cambiado), o el backtracking.

La programación con restricciones puede ser implementada como un lenguaje propio o como bibliotecas para ser usadas en algún lenguaje de programación imperativo. Algunos lenguajes populares de programación con restricciones son [15]:

- B-Prolog (Basado en Prolog, propietario)
- CHIP V5 (Basado en Prolog, también existen bibliotecas en C y C++, propietario)
- Ciao Prolog (Basado en Prolog, software libre: GPL/LGPL)
- ECL^{i}PS^e (Basado en Prolog, software libre)
- Mozart (Basado en Oz, software libre: X11)
- SICStus (Basado en prolog, propietario)
- GNU Prolog (Basado en Prolog, software libre)
- SWI-Prolog Un entorno Prolog que contiene varias librerías para soluciones con restricciones (LGPL)

Algunas bibliotecas populares son:

- Choco (Java, software libre: X11)
- Disolver (C++, propietaria)
- Gecode (C++, software libre: X11)
- ILOG CP (C++,propietaria)
- Koalog Constraint Solver (Java, propietaria)

7.1 Aspecto del Lenguaje ECLiPSe

Con el propósito de validar lo descrito en los capítulos anteriores, se ha desarrollado una herramienta con ECLiPSe, consistente en un solver de distintos problemas, destinado a permitir la ejecución de distintas pruebas [15]. En esta sección se pretende dar una breve introducción a ECLiPSe.

7.2 Lenguaje de Programación ECLiPSe

La forma de resolver los problemas de restricciones es escribir programas especializados, en lenguajes procedurales, que requieren de muchos esfuerzos en su desarrollo, y además se hacen difíciles de mantener, modificar y extender. Siendo cada vez más comunes y eficientes los relacionados a la programación lógica, es por esta razón que ECLiPSe fue escogido, debido a que posee múltiples características que faciliten el desarrollo de un solver CSP, características y especificaciones que hacen a este lenguaje uno de los más apropiados para este tipo de programas [15].

Otras razones fueron:

- Existe abundante información relacionada al uso de ECLiPSe.
- Permite la gestión de archivos.
- Permite el uso de ciclos de repetición, usados en los lenguajes de programación tradicionales.

Además ECLiPSe es un software de código abierto de desarrollo de sistemas y despliegue de aplicaciones de programación con restricciones. Este posee múltiples bibliotecas para solucionar problemas de programación con restricciones, un modelamiento de alto nivel y un control del lenguaje. Es una plataforma lógica de programación altamente declarativa.

El enfoque de Programación Lógica con Restricciones, combina dos paradigmas de programación: Programación lógica y Programación con Restricciones.

- *Programación Lógica*: Formalismo adecuado para la Informática y para la representación del conocimiento. La meta de la programación lógica es probar teoremas representados en lógica de primer orden. Además proporciona una base formal para Prolog.

- *Programación con restricciones*: En este enfoque el proceso de programación está limitado a la generación de restricciones y la solución de estas restricciones.

Las características principales de ECLiPSe es un sistema de Programación Lógica con Restricciones, que consiste en:

- Un núcleo de tiempo de ejecución
- Una colección de Bibliotecas
- Un modelado y control de lenguaje
- Un entorno de desarrollo
- Interfaces para integrarse en un entorno host
- Interfaces para resolucionadores anexos

Uno de los elementos importantes en ECLiPSe, es que provee una forma alternativa de crear ciclos, evitando repeticiones por escrito en forma de predicados recursivos.

- **foreach(X, Lista):** Iterar objetivos con X que abarcan todos los elementos de la Lista. X es una variable local en las metas. También se puede utilizar para la construcción de una lista.

- **for(I, MinExpr, MaxExpr):** Itera objetivos con I que abarcan enteros desde MinExpr a MaxExpr. I es una variable local en objetivos. MinExpr y MaxExpr pueden ser expresiones aritméticas. Puede ser utilizado solamente para control de iteraciones, y su incremento predeterminado es 1.

- **for(I, MinExpr, MaxExpr, Incremen**t**):** Tal como el anterior pero el incremento puede ser especificado.

- **fromto(First, In, Out, Last):** Itera comenzando con In = Fist hasta que Out = Last. In y Out son variables locales.

- **Count (I, Min, Max):** Itera sobre enteros desde Min hasta Max. I es una variable local. Puede ser usada para controlar iteraciones así como también para contar.

Otro de los elementos importantes en ECLiPSe, es que permite el manejo de archivos tanto de entrada como de salida, lo cual facilita de gran manera las distintas pruebas.

Capítulo 8

Autonomous Search

Autonomous Search es una característica de un sistema que tiene la habilidad ventajosa de modificar sus componentes internos cuando se exponga al cambio de fuerzas externas y a oportunidades [16]. Esto corresponde a un caso particular de sistemas adaptativos cuyo objetivo es el de mejorar este rendimiento de resolución mediante su estrategia de búsqueda al problema que se tiene a la mano. Componentes internos corresponden a variados algoritmos involucrados en el proceso de búsqueda-heurísticas, inferencias, etc. Fuerzas externas corresponden a la evolución de la información recogida durante el proceso de búsqueda. Esta información puede ser o bien directamente recogida bajo el problema o directamente computarizada a través de la percepción eficiente de componentes individuales. Ejemplos de información recolectada incluye el tamaño del espacio de búsqueda (exacto o estimado), el número de sub-problemas, etc. La información computarizada incluye el grado de discriminación de heurísticas, la capacidad de poda en técnicas de inferencia, etc. La información puede también referirse al ambiente computacional, el cual puede variar a menudo.

Autonomous Search se puede identificar como un conjunto de Solvers que contienen control en su proceso de resolución, es decir, por un lado considera el ajuste previo a la resolución (ajustes de diferentes componentes del algoritmo antes de intentar resolver una instancia) y por otro lado considera el control durante la resolución (que puede ser lograda ya sea por la modificación de parámetros, heurísticas, funciones de resolución o incluso una función de evaluación).

8.1 Componentes

El simple hecho de elegir una heurística de selección de variable o valor para la resolución del problema de satisfacción de restricciones no otorga el privilegio de encontrar una solución al mismo, y menos, de que ésta sea óptima, por lo tanto es necesario realizar un proceso de resolución necesario para poder evaluar las diferentes estrategias que se poseen que representaran una porción o el camino para encontrar la solución [17].

Para determinar qué alternativa elegir, se necesita de un proceso previo para comenzar la resolución, el que corresponde a un sampling, el cual se encargará de probar cada estrategia de enumeración y guardar una estadística de su comportamiento en base a indicadores, la cual permitirá evaluar, en base a una determinada prioridad, qué estrategia se recomienda para comenzar a resolver el problema.

Para comprender mejor lo anterior, se dará a conocer, cómo se lleva el proceso de resolución. Visto de manera general, mediante el siguiente esquema [18], se tiene lo siguiente:

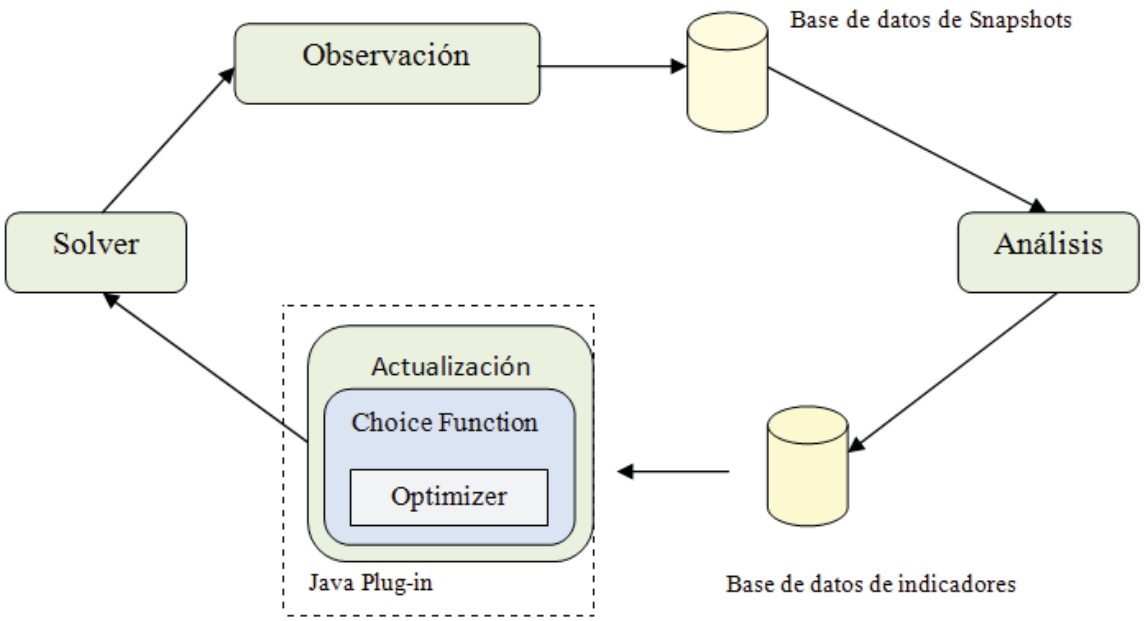


Ilustración 8: Esquema del framework actual.

Los componentes Solver, Observación y Análisis corresponden a procesos que inciden directamente en la resolución. Por otro lado, el componente agregado Actualización incide en la búsqueda de la mejor estrategia de enumeración para enfrentar los problemas y está íntimamente ligado al proceso "Análisis", es decir, es el componente que entrega las prioridades de uso a las mismas. También se puede apreciar por un lado, una base de datos de snapshots y por otro lado una base de datos de indicadores. La información sobre el estado de avance es capturada por medio de dos métodos, siendo uno de ellos el snapshots (captura de pantalla) y el otro por medio de indicadores. Los snapshots son observaciones sobre el actual árbol de búsqueda, en cambio los indicadores son la evidencia de la solución. Dentro de los ejemplos del snapshot se encontraría la máxima profundidad alcanzada en el árbol de búsqueda, la profundidad del nodo actual y el tamaño del actual espacio de búsqueda. Y como ejemplo de los indicadores estarían, la variación de la profundidad máxima.

8.1.1 Solver

La tarea principal de este componente es ejecutar un algoritmo de resolución de CSP. Básicamente, se realiza la búsqueda mediante el algoritmo Forward Checking. Este algoritmo alterna fases de propagación de restricciones y enumeración, además permite encontrar inconsistencias antes de que ocurran (ya que comprueba la consistencia hacia adelante), podando el espacio de búsqueda. Esto reduce su costo.

Para la elección de las "Estrategias" de búsqueda, el Solver deberá preferir la alternativa con mayor prioridad, previamente analizada como la que mejor se comporta al realizar el sampling inicial y, luego, al comparar la estrategia de búsqueda que se está utilizando con las demás estrategias disponibles mientras se ejecuta el Solver. Este también es el encargado de modelar el problema antes de iniciar el proceso de resolución.

8.1.2 Observación

Este componente observa el proceso de resolución y guarda información del estado actual del árbol de búsqueda (Indicadores). Los datos obtenidos se guardan en una base de datos de observaciones (Snapshots). Estas observaciones no son continuas, son "fotografías" de lo que está sucediendo en la búsqueda y pueden ser vistas como una abstracción del estado de resolución en un tiempo t. En este proceso se extrae y guarda información de un estado de resolución.

8.1.3 Análisis

Este componente analiza los datos obtenidos desde las observaciones (Snapshots) tomadas por la "Observación". Se evalúan las distintas estrategias y se provee de indicadores al componente "Actualización". Los indicadores pueden ser extraídos, calculados o deducidos desde una o más observaciones del proceso. Un indicador calculado es el porcentaje de reducción del espacio de búsqueda.

8.1.4 Actualización

Este componente es el más importante, ya que es el encargado de tomar decisiones usando los indicadores. En este componente se interpretan los indicadores y luego se actualizan las prioridades de las estrategias de enumeración a utilizar, además, solicita la ejecución de backtracks al componente Solver. El reemplazo es desarrollado en base a un ranking de calidad, el cual es calculado por medio de la choice function. Éste es responsable de decidir cuál estrategia aplicar a cada paso de decisión durante la búsqueda. Administra un portafolio de estrategias y no tiene conocimientos de ningún problema específico.

8.2 Choice Function

Una choice function se enfoca en capturar la correspondencia entre el rendimiento histórico de cada estrategia de enumeración y el punto de decisión actualmente que se está investigando. Aquí, un punto de decisión o de paso es cada vez que se invoca el solver para fijar una variable de enumeración [8].

La choice function es usada para rankear y elegir entre diferentes estrategias de enumeración en cada paso. Para cualquier estrategia de enumeración S_j, la choice function f en el paso f por cada estrategia S_j está definida por la ecuación 1, donde l es el número de indicadores considerados y a es un parámetro para controlar la relevancia del indicador dentro de la choice function.

$$f_n(S_j) = \sum_{i=1}^{l} \alpha_i \, f_{i_n}(S_j) \tag{1}$$

Adicionalmente, para controlar la relevancia de un indicador i para una estrategia S_j en un período de tiempo, se usa una técnica de estadística popular para producir una serie de tiempo suavizado llamada suavizado exponencial. La idea es asociar, para algunos indicadores, mayor importancia a rendimientos recientes mediante pesos exponencialmente

decrecientes para mayores observaciones. Es este camino, observaciones recientes entregan relativamente más peso que las más antiguas. El suavizado exponencial es aplicado al cálculo de $f_{i_n}(S_j)$, el cual es definido por la ecuación 2 y 3, donde v_0 es el valor del indicador i para la estrategia S_j en el 1, n es un paso dado del proceso, β es el factor suavizado, y $0 < \beta < 1$.

$$f_{i_n}(S_j) = v_0 \tag{2}$$

$$f_{i_n}(S_j) = v_n + \beta_i f_{i_{n-1}}(S_j) \tag{3}$$

Hay que tener en cuenta, que la velocidad a la cual la observación antigua esta suavizada depende de β. Cuando β es cercana a 0, el suavizado es rápido y cuando está cercana a 1, éste es lento.

El procedimiento general de resolución puede ser visto en la Ilustración 9. Tres nuevas llamadas a funciones han sido incluidas: para calcular los indicadores (línea 9), las choice functions (línea 10), y para la elección de prometedoras estrategias (línea 11), esto es, los primeros con mayor choice function. Ellos son llamados después de la propagación con restricciones para calcular los efectos reales de la estrategia (algunos indicadores pueden ser impactados por la propagación). En la línea 2 y 13, el procedimiento para la selección de variables y valores ha sido modificado para responder al reemplazo dinámico de estrategias. Se puede notar, que la selección inicial de la estrategia de enumeración es desarrollada randómicamente fuera del procedimiento.

Los indicadores son los que permitirán el cambio entre una a otra heurística para obtener un mejor resultado en la asignación de las variables. Estos fueron creados en ECL^iPS^e por el grupo de colaboración de optimización de la Pontificia Universidad Católica de Valparaíso. Existen indicadores bases, entre otros: Nodos (N) que registra el número de nodos visitados por un procedimiento, Número de variables no instanciadas (VU) que corresponden al número de variables que aún no se instancian. También existen indicadores que son calculados a partir de los indicadores bases como por ejemplo: Variables instanciadas (VF) que corresponde al número de variables instanciadas por enumeración y propagación, Profundidad (D) que calcula la profundidad actual en el árbol de búsqueda, entre otros.

Procedimiento 2 *solve*(*k* : **integer**, *inst* : **array**) **do**

```
1:  while D[k] ≠ { } and not success do
2:      a ← choose_value_from2(D[k])
3:      inst ← instantiate(inst , k , a)
4:      if consistent(inst , k ,success) then
5:         if success then
6:             print solution(inst)
7:         Else
8:             propagate(k , D ,failure)
```

```
 9:            calculate_indicator()
10:            calculate_choice_functions()
11:            choose_primising_enum_strategy()
12:            if not failure then
13:              l ← choose_variable
14:              solve(l, inst)
15:            end if
16:          end if
17:        end if
18:      restore(k)
19:  end while
```

Ilustración 9: Procedimiento algorítmico para resolver CSPs incluyendo el nuevo enfoque (en comparación al procedimiento 1).

8.3 Optimizadores de parámetros

Con el objeto de determinar el mejor conjunto de parámetros α_i para la choice function se utiliza un enfoque multinivel. Los parámetros son optimizados por un algoritmo genético (GA) que capacita a la choice function llevando a cabo una fase de sampling. El sampling ocurre durante una fase de recolección de información inicial donde la búsqueda es ejecutada repetidamente para fijar el cutoff (cierre), (por ejemplo, bajo un número fijo de variables instanciadas, nodos visitados o backtracks). Después del muestreo, el problema es resuelto con el conjunto de valores más prometedores para la choice function. Hay que considerar que cuando las estrategias tienen el mismo resultado, se selecciona una randómicamente [8].

El algoritmo genético evalúa diferentes combinaciones de parámetros, aliviando la tarea de parametrización manual. Cada miembro de la población codifica los parámetros. Estos individuos son usados para poder crear una instancia de la choice function. Cada choice function instanciada (cada cromosoma) es evaluada en una fase de muestreo, intentando resolver parcialmente el problema para fijar el corte (cutoff). Para evaluar al cromosoma, se utiliza un valor que es entregado por medio de un indicador del proceso de rendimiento (número de backtracks). Luego, cada cromosoma de la población entra en evaluación, selección, crossover y mutación con la finalidad de reproducir una nueva población de choice functions. La choice function resultante es aplicada a resolver el problema CSP.

Se utiliza una población de tamaño 10 y el dominio de parámetros α_i es [-100,100]. El operador crossover randómicamente selecciona dos cromosomas de la población y compañeros de ellos por medio de la selección randómica de un gen y posteriormente intercambia ese gen y todos los demás genes subsecuentes entre los dos cromosomas. Estos son agregados a la lista de cromosomas candidatos. El operador crossover usa un radio fijo, esta operación es desarrollada tantas veces cromosomas hay en la población. El operador de mutación, ejecuta a través de los genes cada uno de los cromosomas de la población y bajo un acuerdo estadístico muta con un radio de 0.1. Estos son agregados a la lista de cromosomas candidatos para la selección natural de procesos.

Capítulo 9

Herramienta

9.1 Implementación

La herramienta que se describirá a continuación, se basa en AS y en ella se raalizarán las futuras modificaciones. Ésta se encarga de resolver problemas específicos de CSP, tales como sudoku, n-reinas, magic square, entre otros, con la característica de que se puede seleccionar el tipo de solución que se quiere obtener, es decir, se debe optar por un tipo random, one strategy o choice function (recordar que esto se relaciona con la manera bajo la cual se desea seleccionar en el transcurso a la estrategia).

9.1.1 Requisitos e instalación de la herramienta

La herramienta sobre la cual está desarrollada la aplicación, es una herramienta que posee un grado de dificultad mínimo a la hora de ejecutarla en un equipo ajeno al lugar bajo el cual está instalada, sólo es importante que el equipo donde se instalará la herramienta posea lo siguiente:

- Como preferencia sistema operativo Windows XP o Windows Vista, ambos de 32 bits.
- Que posea por lo menos 1GB de memoria Ram.

En segundo lugar, se deben tener instalados como mínimo Netbeans IDE 6.8 (o un ambiente similar) y ECLIPSE 6.0. Se le debe agregar a esto, una pequeña modificación con respecto a la ubicación de ECLIPSE 6.0, dado que viene con una ruta especificada por defecto, que eventualmente puede no corresponder con la ruta de ECLIPSE 6.0 en el nuevo equipo (*ver* Ilustración 10).

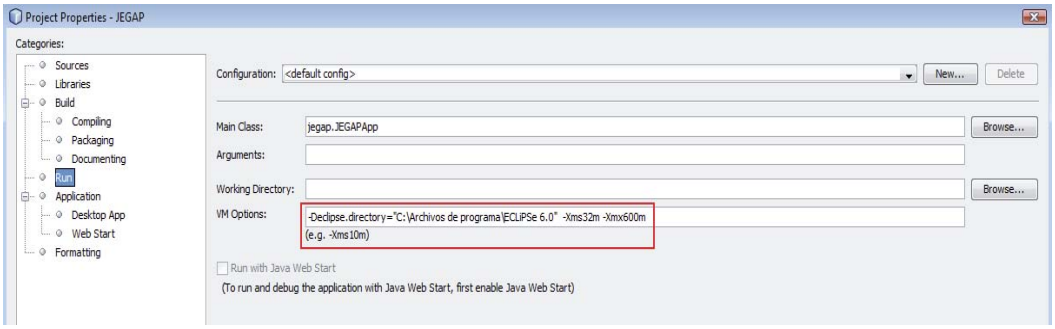


Ilustración 10: Lugar para establecer la ruta de ECLIPSE 6.0 de la herramienta.

9.1.2 Presentación del programa

Al iniciar el funcionamiento de la herramienta, se presenta un menú principal, que corresponde a la clase *JEGAPView.java*. (Ilustración 11)

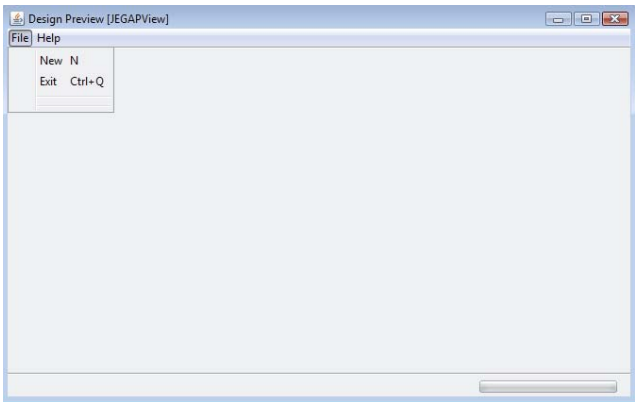


Ilustración 11: Menú principal de la herramienta (Corresponde a la clase JEGAPView.java).

Ésta posee dos opciones (*File*, *Help*), siendo la primera opción la principal, puesto que la segunda son apuntes en relación a la institución a la cual corresponde. En la opción *File* se tiene la alternativa New, la cual permite al usuario ingresar un problema junto a las características de éste, es decir, se debe elegir el problema (Sudoku, n-reinas, entre otros), el tamaño del problema que corresponde a N y finalmente el tipo de solución, que puede ser Random (Ver sección 9.1.2.2), One strategy (Ver sección 9.1.2.3), o Choice Function (Ver sección 9.1.2.4).

9.1.2.1 Análisis de un nuevo problema

Para analizar un nuevo problema se debe seleccionar la opción *New* de la pestaña *File* ubicada en el menú principal, la cual ejecutara la ventana que señala la Ilustración 12.

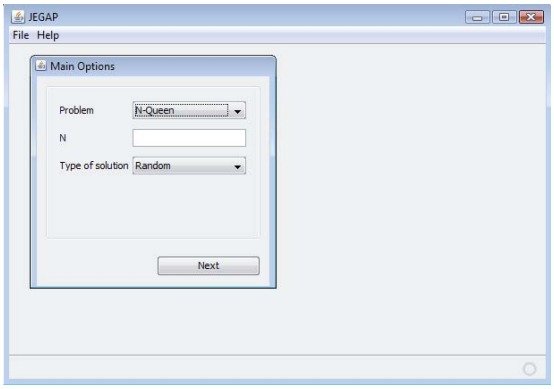


Ilustración 12: Opciones principales requeridas para el análisis de un problema.

Una vez que se especifican las características del problema, se procede con la ejecución de la herramienta, la cual dependerá del tipo de solución seleccionada.

9.1.2.2 Random

En el caso de que el tipo de solución seleccionada sea Random, mostrará la ventana de la Ilustración 13, la cual permite establecer cuáles serán las estrategias que se utilizarán de forma randómica.

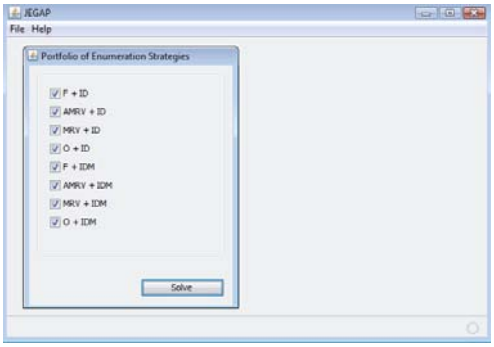


Ilustración 13: Portafolio de estrategias de enumeración para el tipo de solución Random y Choice Function.

Posteriormente, las estrategias que se encuentren etiquetadas, se registrarán en un arreglo con el valor binario *true*, para señalar su etiquetación. Finalizando aparece una barra de espera para señalar que ha comenzado el análisis del problema (ver Ilustración 14).

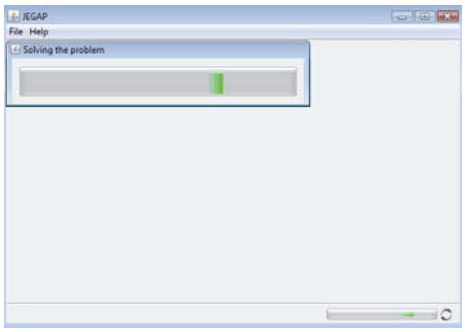


Ilustración 14: Barra de espera que representa el análisis que está realizando la herramienta.

Cabe notar, que la herramienta muestra en su interfaz una barra de espera, indicando con esto que está realizando una serie de pasos:

- Antes de realizar el análisis del problema dado:
  - Establece la comunicación con ECLIPSE 6.0. Se lleva a cabo a través de la función *eclipse.start_and_conexion()*.
  - Establece las variables que representarán al emisor y receptor desde un lado a otro y viceversa, guardando los datos respectivos de acuerdo a la situación. Lo anterior se lleva cabo a través de la función *eclipse. seteo_java()*.
  - La compilación del archivo donde se encuentran codificados los distintos problemas y funciones relacionadas a los mismos. El nombre del archivo es *Eclipse.ed*.

- Luego de lo anterior se procede a realizar lo siguiente:

  - Se establecen las configuraciones relacionadas a las características seleccionadas con anterioridad.
  - Se contabiliza el tiempo que demorará la ejecución.
  - Se resuelve el problema dado. Se selecciona la estrategia Random a utilizar, con el fin de ir resolviendo y obteniendo estados del problema.
  - Se registra el tiempo final que duró el análisis, y los resultados se guardan en un archivo Excel, dentro de la carpeta correspondiente al programa.

9.1.2.3 One Strategy

En caso que el tipo de solución sea One Strategy, mostrará la ventana de la Ilustración 15, la cual permite establecer la estrategia que se utilizará.

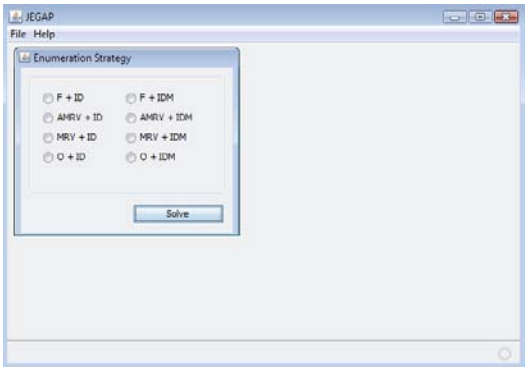


Ilustración 15: Portafolio de estrategias de enumeración para el tipo de solución One Strategy.

Luego de lo anterior, el modo de operar es idéntico al modo en que random lleva a cabo su tarea de análisis, la diferencia está en que la estrategia relacionada ya no es un arreglo sino solamente una sola variable que almacena el nombre de la estrategia.

9.1.2.4 Choice Function

En caso que el tipo de solución seleccionada sea Choice Function, mostrará la ventana de la Ilustración 15, la cual permite establecer el portafolio de estrategias que se podrán utilizar en el transcurso de la búsqueda. De forma interna, el programa registra en una variable binaria llamada *genético* el valor *true*, haciendo alusión a que el paso para el posterior cálculo del algoritmo genético está permitido.

Siguiendo con el procedimiento, se abrirá la ventana de la Ilustración 16. En esta etapa, es importante seleccionar aquellos indicadores que participarán en el análisis.

Luego se mostrará la ventana de la Ilustración 17. En esta sección se debe configurar las características relacionadas al algoritmo genético, en caso que se encuentre activada su posterior utilización.

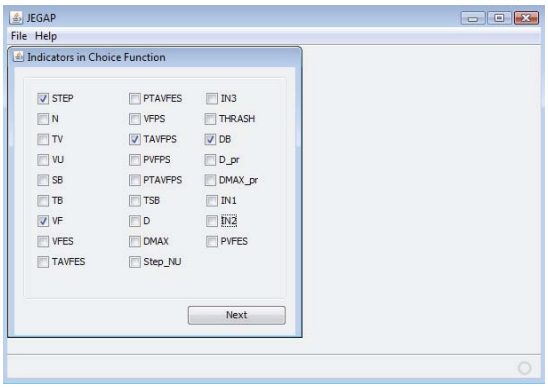


Ilustración 16: Selección de indicadores a utilizar durante el análisis.

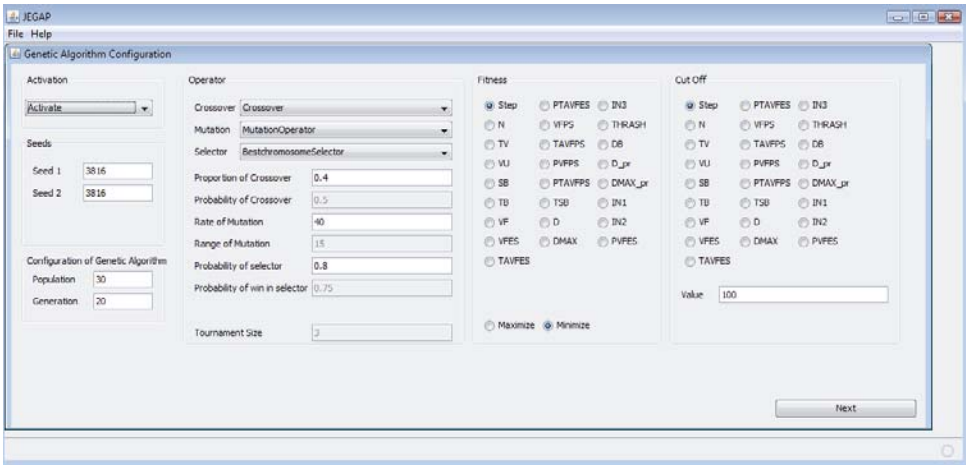


Ilustración 17: Configuración del algoritmo genético.

Si en la configuración anterior el algoritmo genético fue desactivado, se necesita que se ingrese de forma directa los valores de los alfas que se utilizarán en el análisis (ver Ilustración 18). El factor alfa es utilizado para balancear los niveles de exploración entre las diferentes estrategias de enumeración.

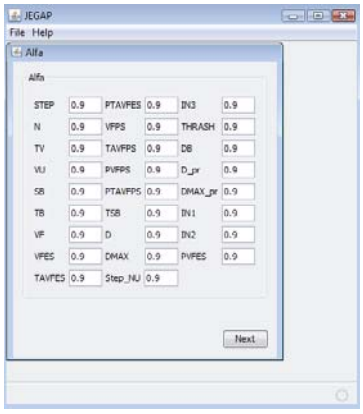


Ilustración 18: Configuración de los alfas a utilizar.

Al contrario, si el algoritmo genético está activado se desplegará la ventana de la Ilustración 19.

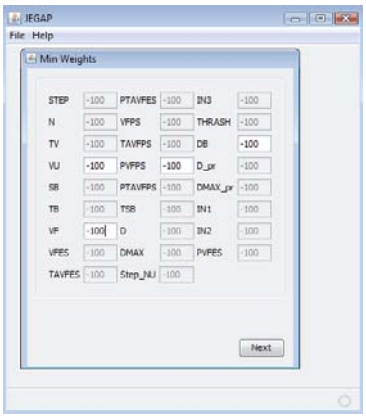


Ilustración 19: Configuración del límite inferior de los alfas a utilizar.

La función que tiene la ventana anterior, es poder registrar el límite inferior de cada uno de los alfas a utilizar. Posteriormente, se necesitara de igual manera su límite superior, para lo cual, una vez ingresado el límite inferior la próxima solicitud correspondiente al límite superior será a través de la ventana de la Ilustración 20.

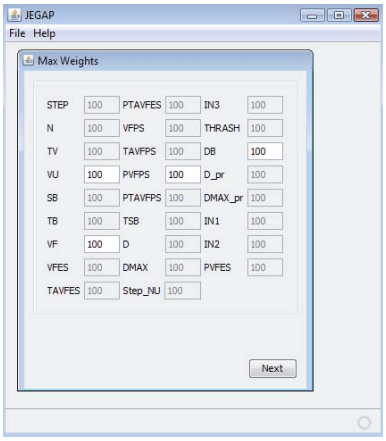


Ilustración 20: Configuración del límite superior de los alfas a utilizar.

Terminada la configuración de los alfa, ya sea a través de la limitación del valor que pueda tomar o el ingreso de forma directa, la siguiente ventana de la Ilustración 21 es la misma para ambos casos y consiste en determinar los factores de suavizado. El factor de suavizado se refiere a la importancia dada al indicador a través del tiempo.

Al terminar los pasos anteriores, el programa visualizará una barra de espera, como se muestra en la Ilustración 14 reflejando el análisis del problema.

De forma interna, se realizan distintos procesos, entre los cuales se destacan los siguientes:

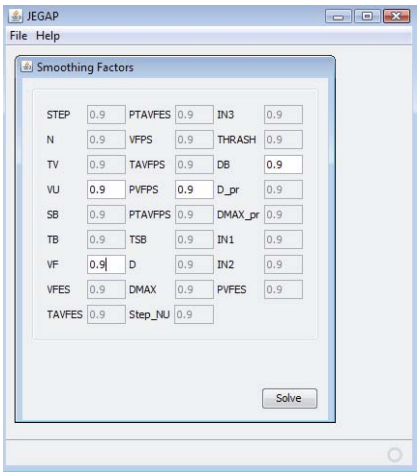


Ilustración 21: Configuración de los factores de suavizado.

- Independiente del camino elegido, entre utilizar algoritmo genético como optimizador y no utilizarlo, se realiza la conexión necesaria entre ECLIPSE 6.0 y Netbeans IDE 6.8.

  Por lo tanto se ejecutan las siguientes tareas:

  - Establecer la conexión con ECLIPSE 6.0. Paso mediante el cual, se lleva a cabo a través de la función *eclipse.start_and_conexion().*
  - Establecer las variables que funcionarán como emisor y receptor desde un lado a otro y viceversa, guardando los datos respectivos de acuerdo a la situación. Paso mediante el cual, se lleva a cabo a través de la función *eclipse. seteo_java().*
  - La compilación del archivo donde se encuentran codificados los distintos problemas y funciones relacionadas a los mismos. El nombre del archivo es *Eclipse.ed*.

- En el caso de que se haya activado el algoritmo genético, se procederá primero a la búsqueda de los valores de los respectivos alfas.

- Luego se procede a realizar lo siguiente:

  - Se establecen ciertas configuraciones relacionadas a las características escogidas con anterioridad.
  - Se comienza a contabilizar el tiempo que demorará la ejecución.
  - Se comienza a resolver el problema dado.
  - Se registra el tiempo final que duró el análisis, y los resultados se guardan en un archivo Excel, en la carpeta correspondiente al programa.

9.2 Corrección de bugs

Los bugs o también llamados errores o defectos del software, son puntos de la herramienta que entorpecen de alguna u otra forma el desarrollo normal de ésta. Sin embargo, cabe destacar el nivel de impacto que ellos tienen durante la ejecución de programa. Los bugs encontrados en el programa, producen un impacto mínimo al momento de ejecutar la herramienta, pero es importante identificarlos para su pronta identificación y reemplazo.

A continuación, se identifican los siguientes bugs encontrados en el programa.

9.2.1 Se desarrolla un problema distinto al seleccionado

- **Descripción**

En uno de los primeros pasos, se debe realizar la selección del problema con el cual se trabajará posteriormente. La Ilustración 22 visualiza el momento en el cual se identifica el problema a resolver.

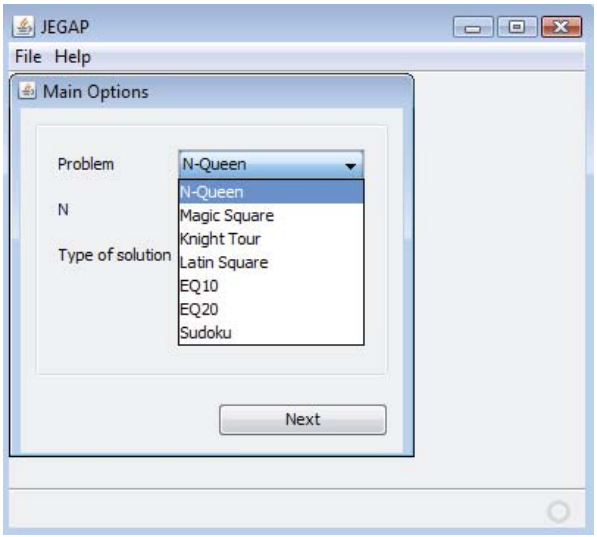


Ilustración 22: Ventana para seleccionar un problema determinado.

- **Problema**

El problema existente, está en el momento de seleccionar algunos modelos que provee la lista asociada al combobox (ver Ilustración 22). Por ejemplo, si el usuario escoge que se resuelva el problema *Eq20*, la herramienta finalmente entrega como resultado, la ejecución del problema *Eq10*. Este problema, tienen relación con la Ilustración 23, la cual muestra de forma detallada el código que permite hacer la asociación entre cada uno de los problemas con su posición en el combobox.

El problema propiamente tal, se encuentra en la mala configuración que poseen algunos *case*. Primero, la numeración de los *case*, se encuentra interrumpida para el *case 2*, y al no existir en el código, lo que hace el programa es que una vez que se elige el problema

Knight Tour en el combobox, al tener la posición 2 se vinculará directamente con el *case 7*, que corresponde al problema sudoku. Dado que *case 2* no existe dentro de los case, se vincula con el case por defecto existente en el código, que en este caso corresponde al *case 7*, resolviendo finalmente la herramienta el problema sudoku en vez del supuestamente seleccionado Knight tour.

Otro error que desencadenará el problema, se debe a la mala vinculación existente entre las posiciones del combobox y la enumeración de case en el código. Por ejemplo, si la elección es *Eq20* (posición 5 en el combobox) se considerará finalmente el *case 5* que corresponde al problema *Eq10*.

- **Solución**

Para solucionar el problema anterior, se debe modificar lo siguiente:

- Se debe sustituir todos los números de los *case*, de forma ascendente, a partir del *case 2*. Debido a que la corrección del número del *case 3* por el *número 2*, el número del case 4 por el número 3, y así sucesivamente producirá el correcto funcionamiento del programa.

9.2.2 Problema en el cálculo del tiempo final del análisis

- **Descripción**

El formato del tiempo final de todo el análisis debe ser congruente con la unidad de medida utilizada, que en este caso corresponde a milisegundos y también debe serlo con la realidad.

- **Problema**

Al finalizar el proceso de análisis, el tiempo registrado en la planilla Excel que el programa genera para mostrar resultados, marca un tiempo que no concuerda con la realidad, puesto que en un caso determinado, el tiempo final mostro el siguiente resultado 1.30400E9m, el cual es un número demasiado grande y no concuerda con el minuto aproximado que duro el análisis en la realidad.

- **Solución**

El problema radica principalmente, en que solo se está midiendo el tiempo final del proceso y no se mide el tiempo inicial del anterior, por lo tanto al hacer la resta entre ambos y obtener el tiempo real transcurrido, sólo se transformaba el tiempo final en milisegundos.

```
166
167         switch(problem.getSelectedIndex()){
168             case 0: //queen
169                 EclipseJava.Indicadores.problema_a_resolver = "queen2("+ n +")";
170                 break;
171             case 1: //magic
172                 EclipseJava.Indicadores.problema_a_resolver = "magic(" + n + ")";
173                 break;
174             case 3:  // kina
175                 EclipseJava.Indicadores.problema_a_resolver = "kina("+ n +")";
176                 break;
177             case 4:  //latin
178                 EclipseJava.Indicadores.problema_a_resolver = "latin("+ n +")";
179                 break;
180             case 5:  //EQ10
181                 EclipseJava.Indicadores.problema_a_resolver = "eq10(10)";
182                 break;
183             case 6:  //EQ20
184                 EclipseJava.Indicadores.problema_a_resolver = "eq20(20)";
185                 break;
186             case 7:  //sudoku
187             default:
188                 EclipseJava.Indicadores.problema_a_resolver = "sudoku("+ n +")";
189                break;
190
191          }
192         //*****************************************************************************
```

Ilustración 23: Switch para asociar cada problema con el lugar que ocupa en el combobox respectivo.

La variable utiliza la función *System.currentTimeMillis()* (ver Ilustración 24), la cual en el momento de mostrar el tiempo transcurrido, sólo estaba convirtiendo la fecha capturada en milisegundos.

```
489     imprimir_resultado(resultado);
490
491      //calculo el tiempo final
492      Indicadores.ind_tiempoFinalmili = System.currentTimeMillis();
493      Indicadores.ind_tiempoFinalnano = System.nanoTime();
494     //calculo el tiempo total
495     Indicadores.ind_tiempoTotalmili = Indicadores.ind_tiempoFinalmili - Indicadores.ind_tiempoIniciomili;
496     Indicadores.ind_tiempoTotalnano = Indicadores.ind_tiempoFinalnano - Indicadores.ind_tiempoInicionano;
497
498
499
500
501     System.out.println("Tiempo Total de Ejecucion es(millis): " + Indicadores.ind_tiempoTotalmili);
502     System.out.println("Tiempo Total de Ejecucion es:((nanos): " + Indicadores.ind_tiempoTotalnano);
503     Indicadores.imprimir_tiempo();
504
505     finalizarTablaExcel();}
506
```

Ilustración 24: Código relacionado con el tiempo total que demora el proceso en realizar el análisis.

9.2.3 Representación errónea del indicador Backtracks

- **Descripción**

Al momento de seleccionar los indicadores que se utilizarán cuando el tipo de solución especificada corresponde a choice function, estos deben corresponder al formato real de cada uno de ellos.

- **Problema**

El problema radica, al momento de seleccionar los indicadores que participarán en el proceso de análisis. El indicador relacionado a Backtracks debería estar representado mediante la sigla B, y al momento de su selección, ocurre el problema de que ésta no se encuentra. (ilustración 25)

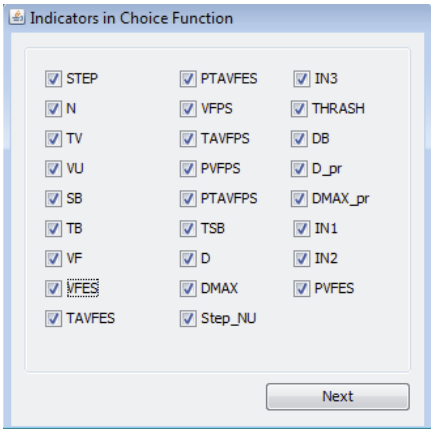


Ilustración 25: Ventana de indicadores.

- **Solución**

Para solucionar el caso anterior, sólo es necesario modificar el nombre de la casilla que posee la sigla TB por B solamente, dado que esa casilla corresponde a Backtracks (ver ilustración 26)

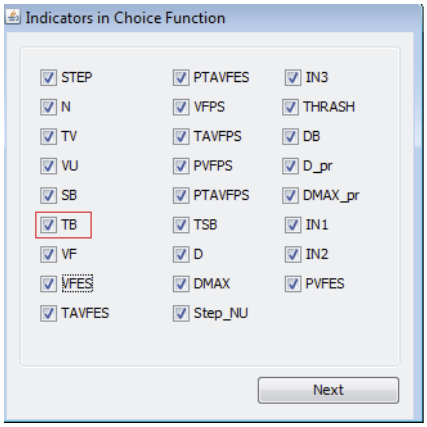


Ilustración 26: Lugar correspondiente a Backtracks (B)

Capítulo 10

Extensión de la implementación

10.1 Ingreso de un nuevo problema

Actualmente, la herramienta no permite el ingreso de un nuevo modelo sino que trabaja bajo los modelos predefinidos en su creación, los cuales se encuentran todos dentro de un mismo archivo y cuyo formato corresponde a un tipo *.ecl*. Es importante mencionar, que un modelo hace referencia a la formalización de la manera en que se va a resolver un tipo de problema, por ejemplo: n-reinas, el cual va acompañado de funciones y determinadas librerías las cuales son necesarias para poder llevar a cabo una compilación y ejecución exitosa al momento de ejecutarla en ECLIPSE 6.0.

10.1.1 Propuesta de inserción de modelo

Dado que la herramienta actual, no posee un mecanismo para que se ingrese un nuevo modelo, se ha utilizado como base la posibilidad de poder particionar el archivo actual que posee todos los modelos y tratar de separarlos de tal manera de que se pueda obtener modelos individuales. Una vez, que se obtienen estos modelos, sólo queda crear el mecanismo que permita la inserción de estos a la herramienta, guardándolos en un lugar específico y a la vez realizando las conexiones necesarias para lograr un buen funcionamiento de su futura ejecución.

10.1.2 Mecanismo de inserción

Para poder llevar a cabo esta tarea, se ha añadido en el menú de presentación del programa, la opción para agregar modelo (ilustración 27).

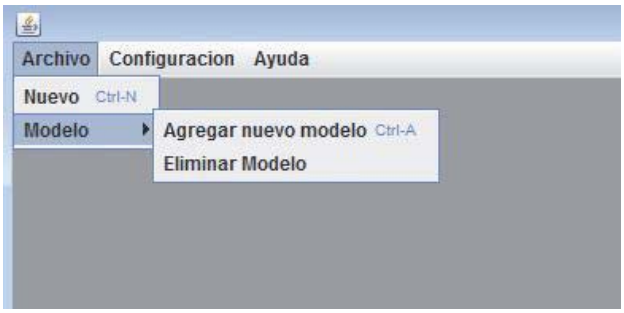


Ilustración 27: Opción para agregar un nuevo modelo a la herramienta.

Una vez, que se selecciona la opción “Agregar modelo”, se dará lugar al despliegue de la ventana de dialogo de la ilustración 28, la cual facilita el ingreso del archivo correspondiente al nuevo modelo, a través de la búsqueda física de éste en el sistema. Esta ventana corresponde a la clase *JFileChooser*, la cual es ofrecida por Netbeans IDE 6.8.

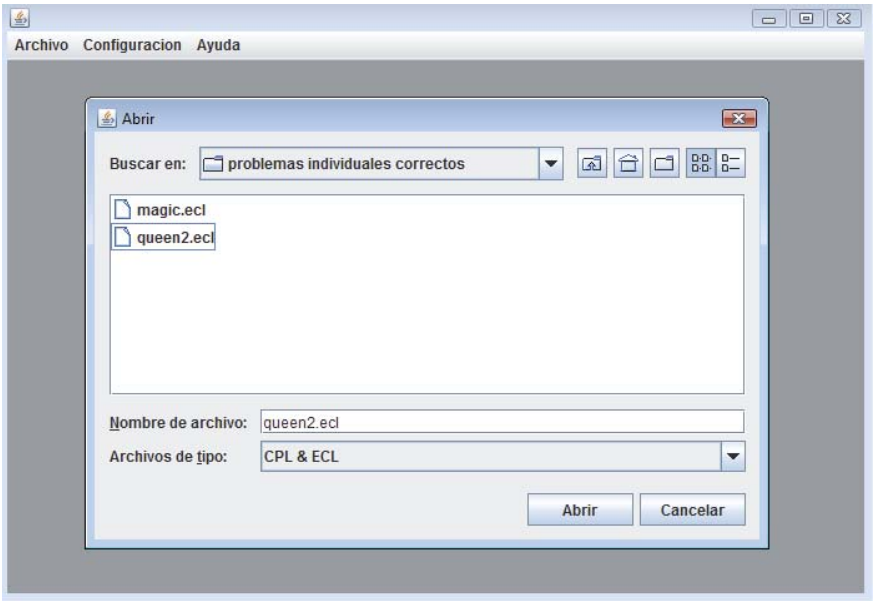


Ilustración 28: Ventana de diálogo para agregar un nuevo modelo a la herramienta.

Al abrir el archivo seleccionado, el programa informa sobre la posible existencia de éste en el programa. En caso que no existan modelos, el programa creará dentro de la carpeta *src* una carpeta cuyo nombre será “problemas”, donde se guardarán todos los posibles nuevos modelos. La Ilustración 29 muestra las dos formas de guardar el modelo, en el lado izquierdo se presenta la forma actual de guardar el único archivo, el cual posee todos los modelos, y en el lado derecho de la imagen se observa la individualización de los respectivos modelos.

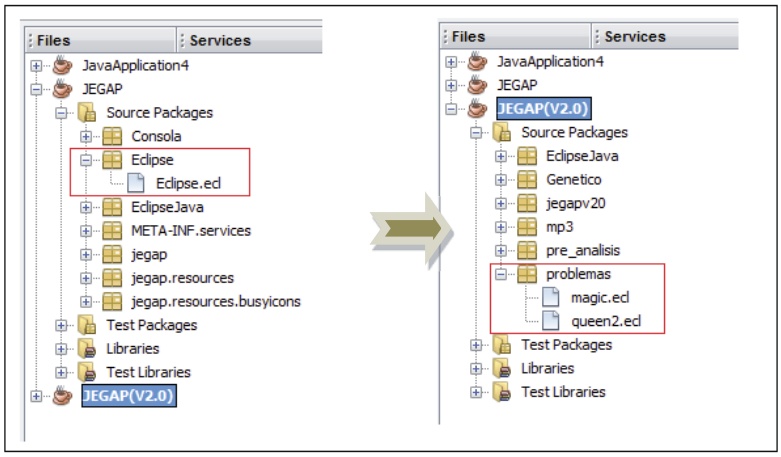


Ilustración 29: Visualización de la forma de guardar modelos. En el lado izquierdo se presenta la forma actual y en el lado derecho se presenta la nueva propuesta.

10.1.3 Librería relacionada a las funciones de Autonomous Search

El archivo perteneciente a los problemas, posee el modelado del problema y todas las funciones relacionadas. Visto esto de manera general, se puede decir, que cada uno de los archivos de los respectivos problemas tendrá todas las funciones de necesita AS en sus respectivos archivos, lo cual puede resultar engorroso en el momento que el usuario quiera modelar un nuevo problema para que pueda ser ejecutado por la herramienta.

Para evitar que los archivos relacionados a los problemas, posean las funciones relacionadas al AS, se creó una librería llamada lib (*lib_autonomous_search*) la cual se debe insertar dentro de las librerías del Solver, que en éste caso corresponde a ECLiPSe.

El usuario, al momento de insertar un nuevo modelo de problema a la herramienta debe considerar lo siguiente:

- La función principal debe llamarse igual que el archivo. Ésta función es la que posee los inputs cuyos valores serán ingresados por el usuario y es la que buscará el programa al momento de ingresar el nuevo problema.
- El modelo del problema a ingresar, debe tener solamente los inputs que rellenará el usuario posteriormente.
- Se debe identificar la función principal que inicia la búsqueda. Generalmente es labeling y ésta debe cambiarse por la función que utilizará AS para su búsqueda personalizada, la cual se llama *lib_autonomous_search().* Además, para que ésta última función sea reconocida, se debe agregar una librería llamada igual que la función.

La librería consta de dos archivos que se visualizan en la Ilustración 30

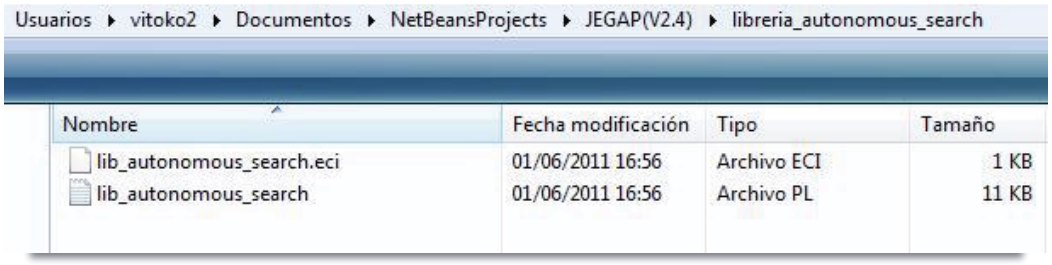


Ilustración 30: Librería Autonomous Search.

Guarda la información relacionada a la exportación que se está realizando, que en éste caso es *lib_autonomous_search*. El archivo con extensión PL posee las funciones relacionadas al AS.

La carpeta de destino, donde se guardarán estos dos archivos, corresponde a la carpeta lib_public del solver ECLiPSe.

La ilustración 31 muestra el problema n-queen modelado para que pueda ser resuelto por la herramienta, y sin la utilización de funciones relacionadas a AS en el mismo archivo.

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


queen(N):-
            queens_lists(N,Board),
            lib_autonomous_search(Board).

queens_lists(N,Board) :-

        length(Board, N),
        Board :: 1..N,

        ( fromto(Board, [Q1|Cols], Cols, []) do
            ( foreach(Q2, Cols), param(Q1), count(Dist,1,_) do
                Q2 #\= Q1,
                Q2 - Q1 #\= Dist,
                Q1 - Q2 #\= Dist
            )
        ).
```

Ilustración 31: Modelo del problema n-queen para ser resuelto por AS mediante la nueva librería.

10.2 Análisis y representación de heurísticas durante el proceso

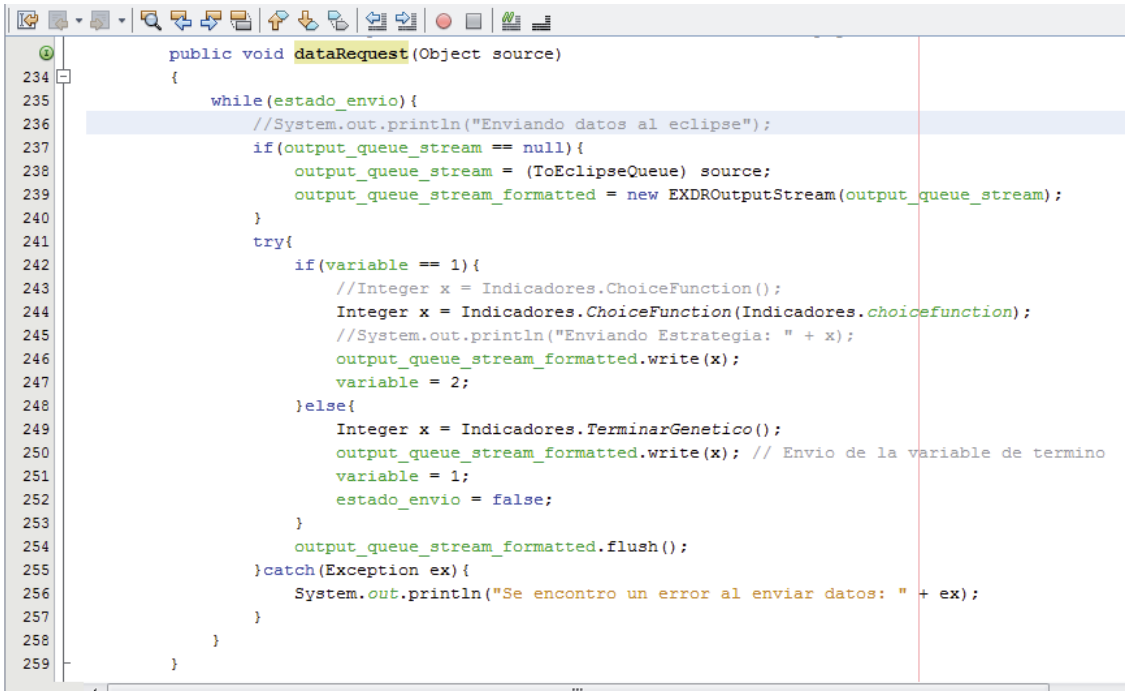
La ejecución del programa, contempla una serie de procesos que son necesarios para lograr el resultado deseado. En primer lugar, de forma general, el programa solicita que se especifiquen las características del programa, por ejemplo, se debe especificar el problema, el tipo de solución, ciertos indicadores, valores de alfa, entre otros. En segundo lugar, una vez que el programa captura la información, comienza su respectivo análisis. Por ejemplo, en el caso de que el tipo de solución relacionada haya sido choice function y se busquen los alfas por determinados intervalos, el programa realizará una pequeña configuración interna, seguida de la búsqueda vinculada a encontrar los valores de alfa. Una vez que se obtienen los valores, se prosigue con el análisis. Posterior a esto, el proceso que viene a continuación está directamente relacionado con los steps que el programa tiene que desarrollar. Un step es similar a un estado de búsqueda y el cual presenta un desarrollo gracias a la utilización de una determinada heurística. Por lo tanto, para analizar preguntas tales como: cuáles fueron las heurísticas utilizadas, cúanto tiempo demoraron éstas, hay que analizar el proceso relacionado a los steps, dentro del programa.

10.2.1 Límites de un step

Durante el desarrollo del step, el programa se encarga de solicitar la heurística adecuada, de acuerdo a las características y desarrollo del proceso, para luego ejecutarla y obtener de esta manera un posible resultado directamente relacionado con la heurística solicitada. En el desarrollo de un step, se pueden realizar otros análisis, como por ejemplo, determinar el número de backtracks realizados hasta ese momento, el número de nodos visitados, entre otras.

En el desarrollo del step, se logra identificar cuando el programa comienza a realizar dos pasos de forma secuencial, los cuales corresponden a la llamada que realiza para obtener una heurística adecuada, la cual corresponde a la clase *EclipseJava.Eclipse.dataRequest()* (ver ilustración 32) y la llegada de datos relacionados a

la heurística dada junto con las características de su ejecución, ésta última corresponde a la clase *EclipseJava.Eclipse.dataAvailable()* (ver ilustración 33).



```
234         public void dataRequest(Object source)
235         {
236             while(estado_envio){
237                 //System.out.println("Enviando datos al eclipse");
238                 if(output_queue_stream == null){
239                     output_queue_stream = (ToEclipseQueue) source;
240                     output_queue_stream_formatted = new EXDROutputStream(output_queue_stream);
241                 }
242                 try{
243                     if(variable == 1){
244                         //Integer x = Indicadores.ChoiceFunction();
245                         Integer x = Indicadores.ChoiceFunction(Indicadores.choicefunction);
246                         //System.out.println("Enviando Estrategia: " + x);
247                         output_queue_stream_formatted.write(x);
248                         variable = 2;
249                     }else{
250                         Integer x = Indicadores.TerminarGenetico();
251                         output_queue_stream_formatted.write(x); // Envio de la variable de termino
252                         variable = 1;
253                         estado_envio = false;
254                     }
255                     output_queue_stream_formatted.flush();
256                 }catch(Exception ex){
257                     System.out.println("Se encontro un error al enviar datos: " + ex);
258                 }
259             }
```

Ilustración 32: Clase dataRequest.

10.2.2 Porcentaje del tiempo utilizado por una heurística

Para conocer el tiempo que demora una determinada heurística, es necesario previamente conocer los límites de ésta. En el punto 10.2.1, se presentaron los límites que posee el step, por lo tanto, la búsqueda para conocer el tiempo que demora un step, se centra en trabajar en el registro dentro de éstas dos clases cuyo funcionamiento secuencial simulan los límites de éste.

Para calcular el porcentaje de una determinada heurística, es necesario relacionarla a un tiempo total. Hasta el momento, el único tiempo total calculado es el que considera la ejecución completa del programa, es decir, desde el momento en que se solicitan los datos relacionados al problema, hasta que se obtiene una solución del mismo. Puesto que el tiempo total, considera también el tiempo utilizado para hacer por ejemplo ciertas configuraciones, la búsqueda de los alfas, entre otros, se optó por considerar y centrar el tiempo en relación al tiempo total que demora la herramienta en ejecutar todos los steps, logrando así acotar el lugar al cual pertenece el porcentaje.

```
// Called when Eclipse flushes source
public void dataAvailable(Object source)
{
    System.out.println("********************** ENTRO A dataAvailable ()    ****************************");
    // la primera vez que se le llama es null luego a contener por ejemplo  com.parctechnologies.eclipse.FromEclipseQueue@142bece
    if(input_queue_stream == null){
        input_queue_stream = (FromEclipseQueue) source;
        input_queue_stream_formatted = new EXDRInputStream(input_queue_stream);
    }
    try{
        CompoundTerm result = (CompoundTerm) input_queue_stream_formatted.readTerm();
        // result vale: com.parctechnologies.eclipse.CompoundTermImpl with [functor=to_java arity=14 arg(1)=0 arg(2)=1 arg(3)=10 arg(4)=9 arg
        System.out.println("Result vale:"+result);
        terminar = result.arg(f_terminar);
        step = result.arg(f_step);
        tv = result.arg(f_tv);
        vu = result.arg(f_vu);
        //ss = result.arg(f_ss);
        sb = result.arg(f_sb);
        b = result.arg(f_b);
        estrategia_usada = result.arg(f_estrategia_usada);
        val = result.arg(f_val);
        var = result.arg(f_var);
        in3 = result.arg(f_in3);
        vfes = result.arg(f_vfes);
        dominio1 = result.arg(f_dominio1);
        resultado = result.arg(f_resultado);
        ssPorRest = result.arg(f_ssPorRest);
        System.out.println(step + " - " + tv + " - " + vu + " - " + sb + " - " + b + " - " + estrategia_usada + " - " + vfes );
        System.out.println("vitoko:"+ terminar+ " - "+ step + " - " + tv + " - " + vu + " - " + sb + " - " + b + " - " + estrategia_usada + "
        if(step != "0"){
            System.out.println("Entro a step!= 0 ");
```

Ilustración 33: Clase dataAvailable.

La ilustración 34 muestra un ejemplo, de la manera en que el programa respalda los tiempos y porcentajes utilizados por cada una de las heurísticas.

Z	AA	AB	AC	AD	AE	AF	AG	AH	AI	AJ	AK	AL
PVFES	***PTAVFE***	***PVFPS***	***PTAVFF***	***Thrash***	***DB***				***Strategy***	***Nanosec***		***% used***
6,25	6,25	0	0	0	0			0	7	3627137	7	0,559210595
6,25	12,5	0	0	1	1			0	6	9327722	6	1,438093177
6,25	18,75	0	0	1	1			0	3	534146	3	0,082351481
6,25	25	0	0	2	1			0	3	536312	3	0,082685422
6,25	31,25	0	0	3	1			0	3	352349	3	0,054323091
6,25	37,5	6,25	6,25	4	1			0	3	895924	3	0,13812828
6,25	43,75	6,25	12,5	5	2			0	3	3546400	3	0,546763041
6,25	50	0	12,5	5	5			0	3	3831842	3	0,590770805
6,25	56,25	6,25	18,75	4	5			0	3	1106425	3	0,170582082
6,25	62,5	6,25	25	5	6			0	3	1982794	3	0,305695487
6,25	68,75	0	25	5	9			0	3	2891429	3	0,445783474
6,25	75	0	25	4	11			0	3	4410826	3	0,680035144
6,25	81,25	0	25	3	11			0	3	531143	3	0,081888496
6,25	87,5	6,25	31,25	4	11			0	3	1293251	3	0,199385814
6,25	93,75	6,25	37,5	5	12			0	3	2433200	3	0,375136429
6,25	100	0	37,5	5	15			0	3	2476921	3	0,381877075
6,25	106,25	6,25	43,75	4	15			0	3	1171797	3	0,180660752
6,25	112,5	6,25	50	5	16			0	3	1376851	3	0,212274769
6,25	118,75	0	50	5	19			0	3	2052147	3	0,316387924
6,25	125	0	50	4	21			0	3	4156254	3	0,640786734
6,25	131,25	0	50	3	21			0	3	526463	3	0,081166961
6,25	137,5	6,25	56,25	4	21			0	3	1589518	3	0,245062513
6,25	143,75	6,25	62,5	5	22			0	3	1408629	3	0,217174113
6,25	150	0	62,5	5	25			0	3	2078546	3	0,320457966
6,25	156,25	0	62,5	4	27			0	3	4568108	3	0,704283956
6,25	162,5	0	62,5	3	27			0	3	426940	3	0,065823092
6,25	168,75	0	62,5	4	30			0	3	8101448	3	1,249033483
6,25	175	0	62,5	2	30			0	3	440908	3	0,067976596
6,25	181,25	0	62,5	3	30			0	3	353746	3	0,054538472
6,25	187,5	6,25	68,75	4	30			0	3	959410	3	0,147916177
6,25	193,75	6,25	75	5	31			0	3	2373486	3	0,365930076

Ilustración 34: Respaldo del tiempo y porcentaje del tiempo utilizado por cada heurística en la planilla Excel que entrega el programa.

10.2.3 Tiempo de la secuencia de heurísticas en término de porcentajes

De manera de profundizar los conocimientos que se pueden obtener a partir del punto 10.2.2, se ha establecido registrar la secuencia de las heurísticas que participaron en el proceso, es decir, ya no se consideran las heurísticas de forma individual, sino que lo relevante es considerar el porcentaje de tiempo utilizado por la heurística antes de cambiarse a otra durante la ejecución del proceso.

Para lo anterior, se consideró la idea de frecuencia acumulada, El mecanismo de análisis se basa en la tabla de resultados de la ilustración 32, siendo la idea principal identificar la heurística determinada e ir acumulando el tiempo de ésta, hasta que sea reemplazada por una nueva heurística. La ilustración 35 muestra la tabla relacionada a la secuencia de heurísticas obtenida, cabe recordar que ésta tabla también se adjunta en el archivo Excel que el programa entrega una vez finalizado el proceso de búsqueda.

AC	AD	AE	AF	AG	AH	AI	AJ	AK	AL	AM	AN	AO	AP
PTAVFF	*Thrash*	*DB*				*Strategy*	*Nanosec*		*% used*		*Accumulated Nano*		*% Accumulated Nano*
0	0	0			0	7	3627137	7	0,559210595	7	3627137	7	0,559210595
0	1	1			0	6	9327722	6	1,438093177	6	9327722	6	1,438093177
0	1	1			0	3	534146	3	0,082351481	3	635662501	3	98,00269623
0	2	1			0	3	536312	3	0,082685422				
0	3	1			0	3	352349	3	0,054323091				
6,25	4	1			0	3	895924	3	0,13812828				
12,5	5	2			0	3	3546400	3	0,546763041				
12,5	5	5			0	3	3831842	3	0,590770805				
18,75	4	5			0	3	1106425	3	0,170582082				
25	5	6			0	3	1982794	3	0,305695487				
25	5	9			0	3	2891429	3	0,445783474				
25	4	11			0	3	4410826	3	0,680035144				
25	3	11			0	3	531143	3	0,081888496				
31,25	4	11			0	3	1293251	3	0,199385814				
37,5	5	12			0	3	2433200	3	0,375136429				
37,5	5	15			0	3	2476921	3	0,381877075				
43,75	4	15			0	3	1171797	3	0,180660752				
50	5	16			0	3	1376851	3	0,212274769				
50	5	19			0	3	2052147	3	0,316387924				
50	4	21			0	3	4156254	3	0,640786734				
50	3	21			0	3	526463	3	0,081166961				
56,25	4	21			0	3	1589518	3	0,245062513				
62,5	5	22			0	3	1408629	3	0,217174113				
62,5	5	25			0	3	2078546	3	0,320457966				
62,5	4	27			0	3	4568108	3	0,704283956				
62,5	3	27			0	3	426940	3	0,065823092				
62,5	4	30			0	3	8101448	3	1,249033483				
62,5	2	30			0	3	440908	3	0,067976596				
62,5	3	30			0	3	353746	3	0,054538472				
68,75	4	30			0	3	959410	3	0,147916177				

Ilustración 35: Respaldo del tiempo y porcentaje del tiempo acumulado por cada heurística en la planilla Excel que entrega el programa.

10.2.4 Generador de gráficos

La herramienta actual, no emite ningún tipo de resultado que sea representado a través de gráficos. Esta representación es nueva, y es complementaria a la información numérica de la cual se dispone. El tipo de gráfico seleccionado, ha sido el gráfico de barras dado que pretende resaltar la representación de porcentajes de datos que componen un total. Para lograr el diseño de un gráfico de barras, se utilizó una librería gratuita llamada *JFreeChart.*

- **Representación gráfica del porcentaje del tiempo utilizado por cada heurística durante el proceso**

El punto 10.2.2, trata la forma bajo la cual la herramienta calcula el porcentaje del tiempo de participación de una heurística determinada. Además, esta información numérica es utilizada como base para la construcción del gráfico. Los datos utilizados, corresponden a las dos columnas que guardan datos relacionados a este proceso, es decir, se considera la columna AK y AL de la ilustración 34. La columna AK guarda las heurísticas utilizadas en cada step y la columna AL guarda el porcentaje de tiempo utilizado por cada una. La columna AL se utiliza para relacionar cada barra del gráfico con la heurística utilizada (estos números aparecen en eje x del gráfico), mientras que la columna AL se utiliza para mostrar rangos dentro de los cuales varía el porcentaje de cada barra (estos números aparecen en eje y del gráfico). La ilustración 36 visualiza el gráfico señalado anteriormente.

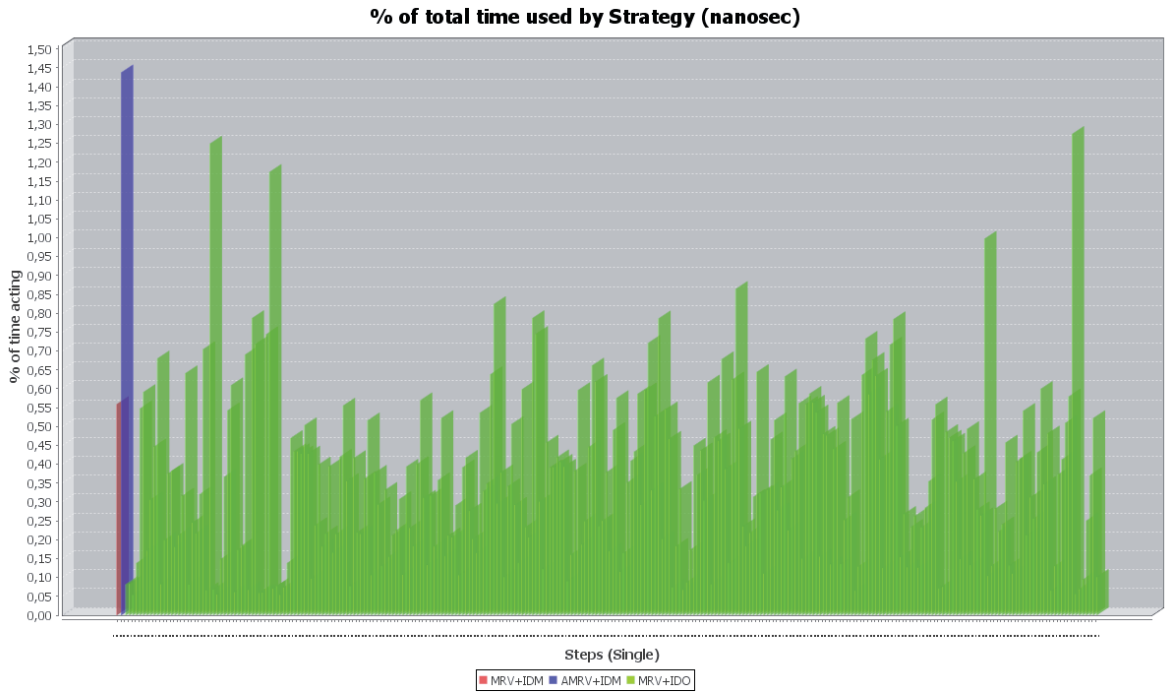


Ilustración 36: Representación gráfica del porcentaje del tiempo utilizado por cada heurística.

- **Representación gráfica del tiempo relacionado a la secuencia de heurísticas en términos de porcentaje**

El punto 10.2.3, explica la manera de cómo la herramienta calcula el tiempo relacionado a la secuencia de heurísticas en términos de porcentaje. Los datos utilizados, corresponden a las dos columnas que guardan datos relacionados a este proceso, es decir, se considera la columna AO y AP de la Ilustración 35. La columna AO guarda las heurísticas utilizadas en cada step y la columna AP guarda el porcentaje de tiempo acumulado utilizado por cada secuencia. La columna AO se utiliza para relacionar cada barra del gráfico con la heurística utilizada (estos números aparecen en eje x del gráfico), mientras que la columna AP se utiliza para mostrar rangos dentro de los cuales varia el porcentaje de

cada barra (estos números aparecen en eje y del gráfico). La ilustración 37 visualiza el gráfico señalado anteriormente.

10.2.4.1 Sustitución del tiempo total considerado para obtener el porcentaje de participación de los steps.

Para obtener el porcentaje de participación de cada step durante la ejecución, éste se calcula en relación a un tiempo total, el cual corresponde al tiempo total consumido por el proceso denominado ECLiPSe (tiempo que referencia el tiempo total por parte de los steps).

Para tener una mejor visualización de los datos se recomendó considerar como tiempo total, el tiempo que demora la herramienta desde que comienza a ejecutar el algoritmo genético hasta que obtiene el resultado. Por lo tanto, el tiempo total aumentará significativamente, dado que ahora se considerara el impacto que genera el algoritmo genético en el resultado.

10.2.4.2 Corrección de la unidad de tiempo considerada.

Lo visto en el punto 10.2.4.1, afectó la visualización de las barras correspondientes a steps del gráfico de la hoja de resultados del archivo Excel, dado que se considerará un tiempo total mucho mayor, lo que provocó que las barras disminuyeran su tamaño considerablemente, haciendo la mayoría de los casos invisibles de visualizar. Para ello, se modificó la unidad de tiempo de milisegundos a nanosegundos, lo cual generó, una mejora de la visualización de las barras en el gráfico.

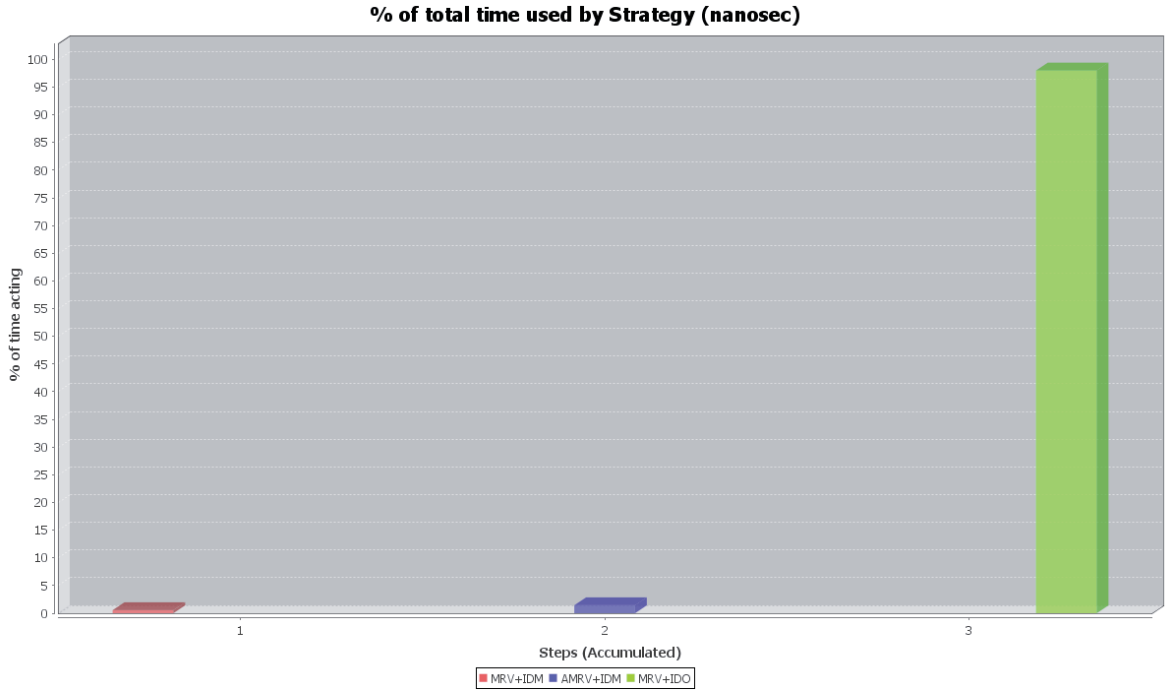


Ilustración 37: Representación gráfica del tiempo relacionado a la secuencia de heurísticas.

10.3 Rediseño de la interfaz gráfica

La interfaz gráfica actual que posee la herramienta, contiene varias funcionalidades relacionadas con las tareas necesarias para llevar a cabo el desarrollo y el resultado de acuerdo a las necesidades de ese entonces. En tiempos de ahora, las expectativas son superiores y es necesario el diseño de una interfaz gráfica que mejore y beneficie al usuario, en el momento durante el cual se lleva a cabo el análisis.

La gráfica total del programa ha sido renovada y los principales cambios que se han realizado en la interfaz gráfica, son los siguientes:

10.3.1 Aumento de la cantidad consecutiva de problemas a desarrollar

Actualmente, la herramienta sólo puede realizar una prueba de forma consecutiva, es decir, el usuario debe esperar a que el problema termine para comenzar nuevamente y seleccionar las características de éste. Sin embargo, este problema se ve con claridad al momento en que el usuario desea realizar varias pruebas a un mismo problema, con las mismas características relacionadas y de forma consecutiva, puesto que debe ingresar los mismos datos cada vez que finaliza uno de ellos.

Para mejorar lo anterior, se desarrolló una nueva interfaz que permite ingresar la cantidad de problemas que se quiere desarrollar. Estos cambios se realizaron en dos partes:

- La primera interfaz que renovó su gráfica, fue la de la Ilustración 36. Corresponde a una de las primeras solicitudes que contempla características relacionadas al problema. En ella, se puede visualizar, que dentro de las solicitudes normales, ahora se adjunta la cantidad de problemas a desarrollar.

- Se agregó una nueva interfaz, que contempla de manera visual el avance del análisis de cada problema. Por ejemplo, si el usuario desea desarrollar seis problemas, la interfaz permite que el usuario pueda inspeccionar que problema se ha analizado, cuál se está analizando, y cuál falta por analizar. Lo anterior, se relaciona con la ilustración 39.

Cabe mencionar, que una vez que finalizan todas las pruebas, aparece en la esquina superior derecha una pequeña ayuda, para poder abrir cualquiera de las pruebas de forma inmediata y sin la necesidad de recurrir a abrir el archivo a través de la búsqueda manual de la ruta de éste. La ilustración 40 visualiza la ayuda anterior.

10.3.2 Nueva pestaña de Configuración

En la herramienta actual, cada vez que el usuario solicita como optimizador el algoritmo genético, debe configurar las características del optimizador, del fitness[2] y del cutoff[3].

A partir de lo anterior, se retiró la ventana que solicita la configuración de las características antes mencionadas, en su lugar el usuario sólo deberá escoger el optimizador determinado (ver ilustración 41) y realizar la configuración previa, aunque posee la configuración más utilizada por defecto.

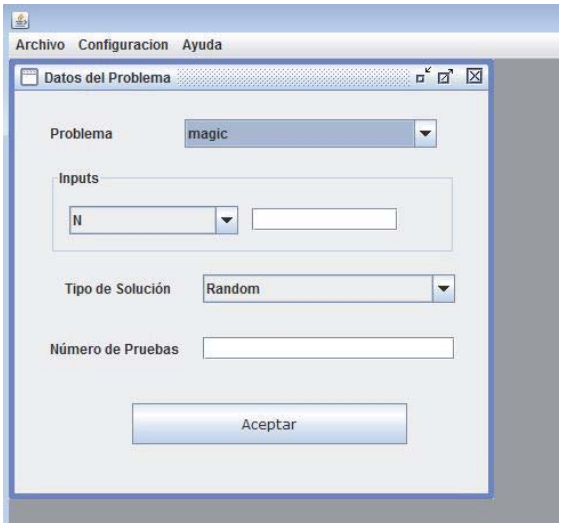


Ilustración 38: Ventana que posee la nueva opción para agregar un determinado número de pruebas.

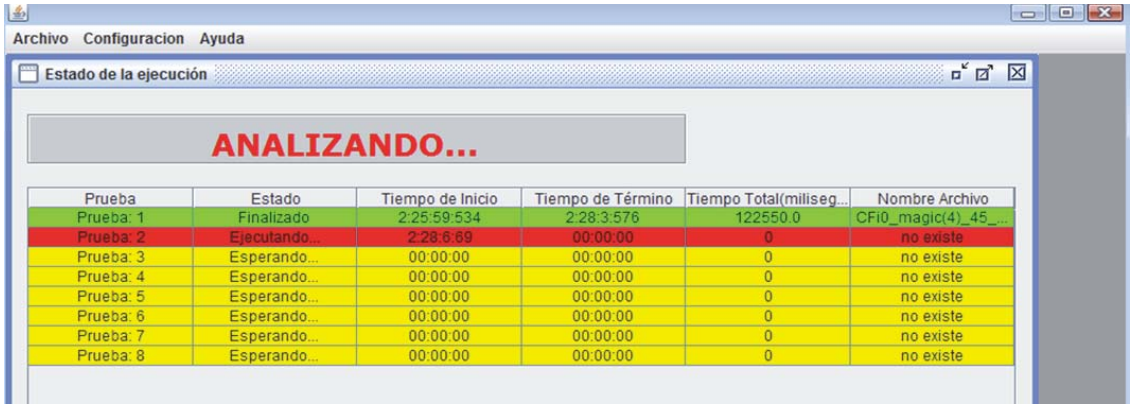


Ilustración 39: Ventana que identifica qué pruebas se han realizado, cuál se está realizando y cuáles faltan por analizar.

De acuerdo a la ilustración 42, la pestaña configuración despliega las siguientes opciones:

[2] El fitness es la función de evaluación que indica qué tan bueno es el individuo (es decir, la solución del problema) con respecto a los demás.
[3] Se relaciona al tiempo en que debería de terminar tal ejecución.

- **Choice Function**

La opción Choice Function, corresponde a que el usuario tenga la posibilidad de configurar los optimizadores a utilizar. En estos momentos, el único optimizador que puede configurarse es el algoritmo genético, cuyas opciones se pueden ver en la ilustración 43.

- **Fitness**

Posee las opciones relacionadas con la configuración del fitness, las cuales se muestran en la ilustración 44.

- **Cut Off**

Posee las opciones relacionadas con la configuración del Cut off, las cuales se muestran en la ilustración 45.

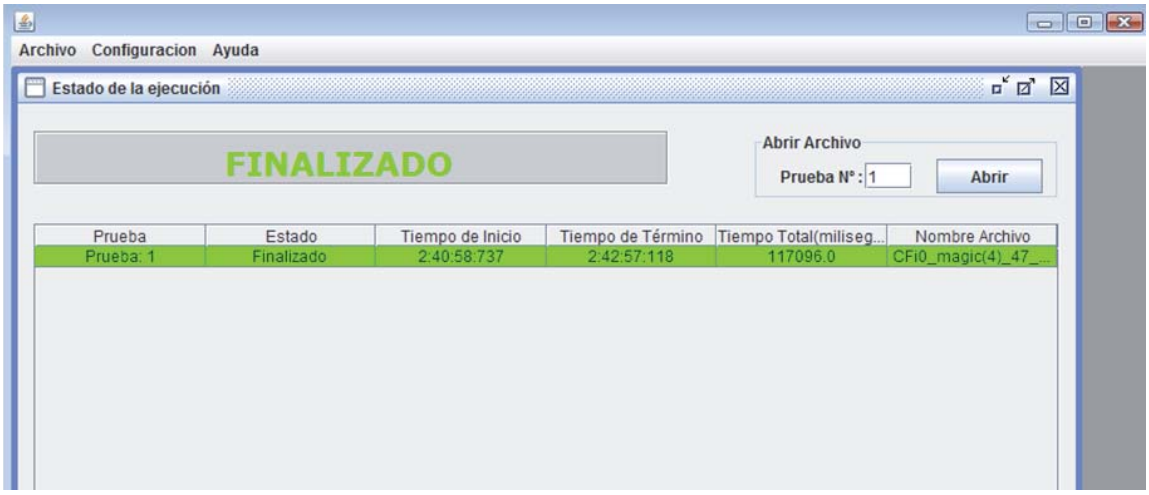


Ilustración 40: Visualización de la opción para abrir archivo Excel de forma directa, una vez finalizada todas las pruebas.

10.4 Integración de dos nuevos problemas

La integración de nuevos problemas, en general, es un paso de adaptación que requiere de ciertas modificaciones para que pueda ser ejecutado correctamente y de forma eficaz por la herramienta. Estas modificaciones, en teoría, son similares en la mayoría de los problemas integrados, aunque dependen de la forma bajo la cual fueron programados. Sin embargo, uno de los problemas nuevos difiere de todos los anteriores, dado que corresponde a un problema de optimización de restricciones (Constrained optimisation problems - COP).

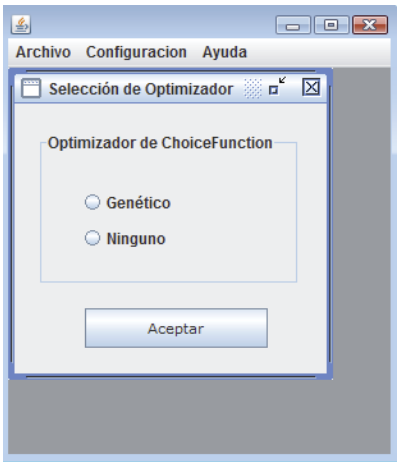


Ilustración 41: Ventana para seleccionar el optimizador de Choice Function.

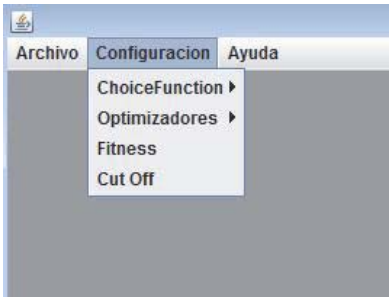


Ilustración 42: Pestaña configuración.

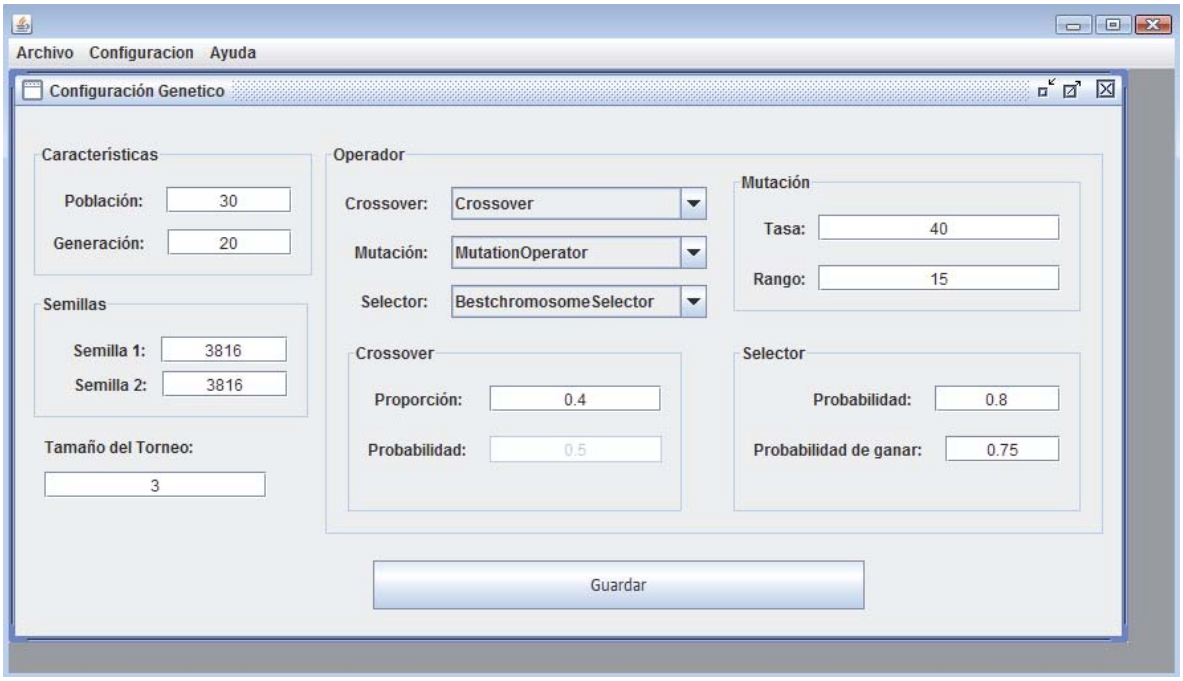


Ilustración 43: Configuración del algoritmo genético.

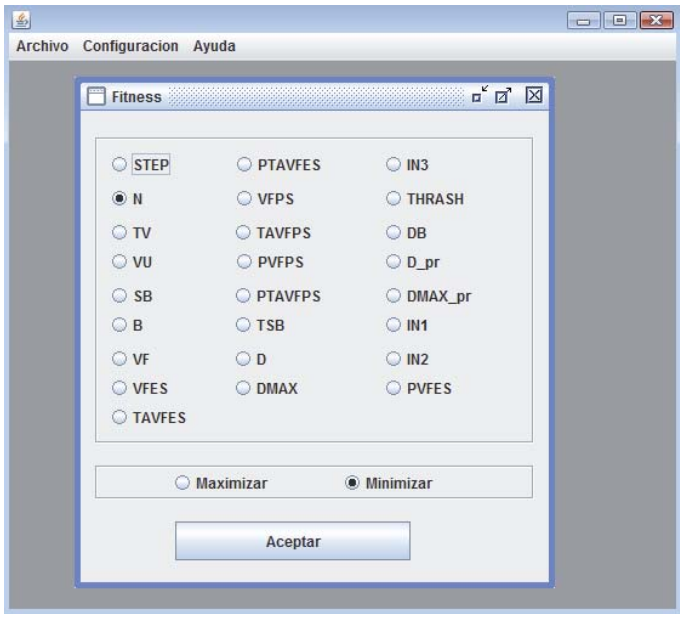


Ilustración 44: Configuración del Fitness.

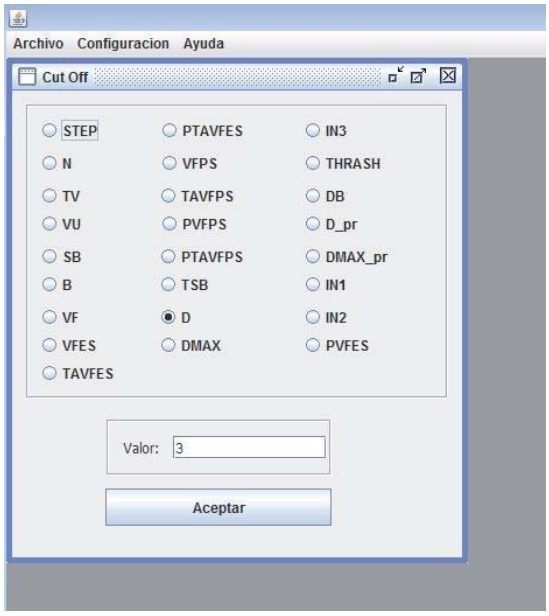


Ilustración 45: Configuración del Cut Off.

10.4.1 NRP (Nurses Rostering Problem)

El NRP corresponde a producir una lista de asignaciones de turnos (ya sea semanal, quincenal o mensual) para el personal de enfermería de un determinado centro sujeto a una variedad de restricciones que pueden ser específicos de cada hospital.

Este problema corresponde a un problema de satisfacción de restricciones, por lo tanto las modificaciones que se deben realizar, deben ser similares a los cambios que se han

hecho a los problemas ya integrados en la herramienta. El código fuente del problema, se encuentra en la sección anexos de éste documento.

Antes de comenzar a realizar los respectivos cambios, se debe identificar tres partes importantes que componen un problema de satisfacción de restricciones que son: Variables, Restricciones y la Búsqueda respectiva. Dado que las modificaciones, en el modelo, se realizan directamente en la manera de realizar la búsqueda de éste y dejando variables y restricciones sin modificación.

Para el caso en observación, la función `solve` es la que comienza a realizar la respectiva búsqueda y la variable `List_C` es la lista encargada de acumular el resultado finalmente obtenido. Por lo tanto se harán las siguientes modificaciones:

- `nrp_copia(Nurses,Paramedics,C_Nurses):-`

```
nrp2(Nurses,Paramedics,C_Nurses,Board),
Board2 = Board,
init_globals,
seTssTotal(Board),
lab(Board,Board2),
end_java(Board).
```

Donde `nrp2` corresponde a la función `nrp` del código anterior, pero con la diferencia de que `Board` corresponderá a `List_C`.

- `nrp2(Nurses,Paramedics,C_Nurses,List_C)`

Como se explicó anteriormente `nrp2` corresponde a la función `nrp` del código anterior. La modificación que se realizó, es enviar la variable `List_C` para que realice la búsqueda la función `lab(Board,Board2)`. Todas las demás afirmaciones, que contiene `nrp2` tienen que ir obligatoriamente para que se realice correctamente la resolución del problema.

El resultado obtenido en ECLiPSe, corresponde a tres matrices (ilustración 46, 47 y 48) y a diferencia del resultado obtenido por la herramienta Autonomous Search, es que se obtiene una sola lista que corresponde a la suma total de los elementos de las tres matrices obtenidas en ECLiPSe.

10.4.2 MCDP (Manufacturing cell design problem)

El Problema de Formación de Células de Manufactura considera la siguiente situación. Dado un conjunto de máquinas, un conjunto de partes e información referente a las máquinas necesarias para procesar cada una de las partes, se requiere encontrar una partición del conjunto de máquinas en células de manufactura y el conjunto de partes en familias de partes, de manera tal que se minimice el movimiento intercelular.

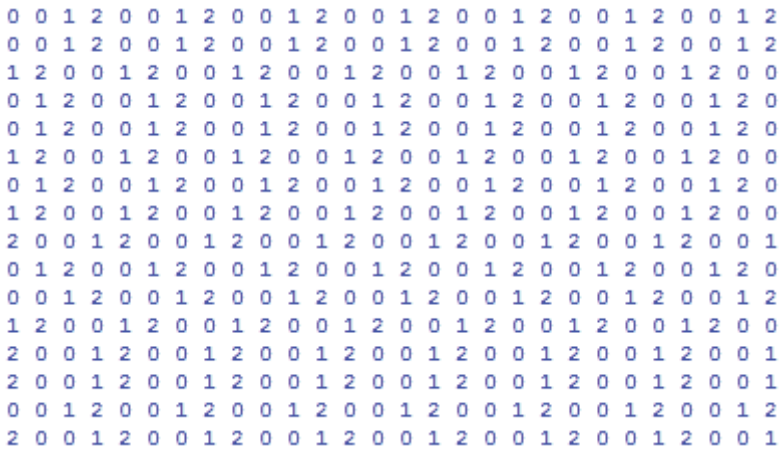


Ilustración 46: Corresponde a la matriz Schedule.

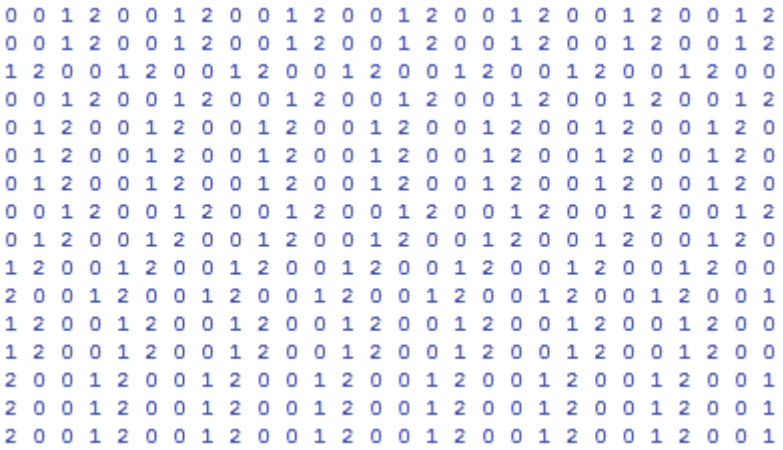


Ilustración 47: Corresponde a la matriz Schedule_Param.

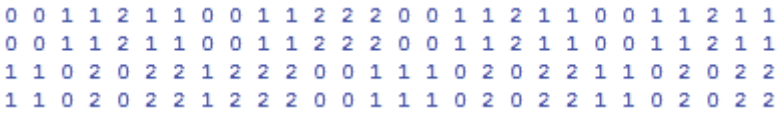


Ilustración 48: Corresponde a la matriz Schedule_Chief.

Este es un problema de optimización de restricciones (COP), por lo tanto las modificaciones que se deben realizar, constituye un nuevo desafío dado que no se posee algún problema similar a éste. El código fuente del problema, se encuentra en la sección anexos de éste documento.

De la misma forma que el problema anterior, la idea principal es separar del modelo las variables y restricciones del proceso de búsqueda. A diferencia del problema anterior, aquí se identifica una función de costo, la cual constituye el elemento principal de un COP.

ECLiPSe nos permite resolver mediante la combinación de la propagación con restricciones con diversas formas del método de ramificación y acotamiento. Esta forma de búsqueda no necesita ser explícitamente programada ya que necesita ser reconocida por la librería `branch_and_bound`.

Para el caso en observación, se realizaron varias modificaciones para poder obtener una lista cuya solución era correcta. De todas las modificaciones, realizadas sólo se pudo obtener una solución al momento de elegir la función `term_variables(Board,Vars)` como función de búsqueda y hacerles las modificaciones que se realizaron en el problema para su adaptación. Lamentablemente, el resultado obtenido al considerar `term_variables(Board,Vars)` no es equivalente con la solución obtenida en eclipse. Mediante ECLiPSe se obtiene el resultado de la Ilustración 49:

```
[machines : 5, cells : 3, parts : 7, mmax : 4, sum : 1]
Found a solution with cost 4
Found a solution with cost 3
Found no solution with cost 0.0 .. 2.0
total_cost : 3
total_cost2 : 3
backtracks : 1
Machines:
assigned_machines : [](2, 1, 1, 1, 1)

Parts:
assigned_parts : [](1, 1, 1, 1, 1, 1, 2)
```

Ilustración 49: Resultado del problema MCDP obtenido mediante ECLiPSe.

Siendo el resultado obtenido por la adaptación mediante Autonomous Search equivalente a [3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 2] lo cual no corresponde con lo obtenido en ECLiPSe.

Hubo modificaciones constantes en el `minimize` puesto que éste ejecuta la búsqueda de ramificación y acotamiento, pero no hubo resultados en ninguno de los intentos realizados.

10.5 Ingreso de un nuevo indicador

Hasta el momento, sólo se cuentan con indicadores establecidos, es decir no tiene la posibilidad de que se ingresen nuevos indicadores. Las configuraciones que se deben realizar en relación a los indicadores, se encuentran desde el momento en que el usuario comienza a seleccionar éstos en adelante, siendo la mayor cantidad de éstas llevadas a cabo en la conexión que se realiza entre ECLiPSe y Autonomous Search.

10.5.1 Enfoque que utiliza Autonomous Search para trabajar con los indicadores.

La idea central que se ocupa en Autonomous Search para poder relacionar los datos que elige el usuario, es trabajar con una matriz cuyas dimensiones son:

- Las columnas de la matriz, están conformadas por una serie de indicadores que van de forma obligatoria y el conjunto de indicadores que el usuario puede seleccionar.
- La cantidad de filas que posee son 10 fijas.

Existen filas importantes dentro del proceso, como lo es “enable” la cual guarda aquellos indicadores que selecciona el usuario. También existen otras filas, como por ejemplo grmax y grmin que guarda los límites mínimos y máximos seleccionados por el usuario para alfa para un problema de tipo choice function. Todo lo anterior, lleva lugar en la selección necesaria por parte del usuario.

Durante el proceso de envío de datos entre ECLiPSe y AS se debe establecer una serie de restricciones con respecto a los indicadores existentes. Por ejemplo, para la actualización de los indicadores, durante el proceso de envío, se fijan funciones de acuerdo a los indicadores establecidos, cuyo código se puede visualizar en la ilustración 50. En esta figura, para actualizar el valor de un indicador compuesto se requieren de otros a la vez.

Finalmente las últimas configuraciones que existen en relación a los indicadores, están escritas de forma explícita en el problema mismo. Es decir, no en el modelo para resolver el problema, sino que en las funciones que son necesarias para adaptar el problema a AS. De manera similar a algunas configuraciones necesarias para poder realizar el envío de datos entre ECLiPSe y Autonomous Search, en ECLiPSe se deben establecer la forma en que se recibirán esos datos. La ilustración 51 visualiza una configuración necesaria en el problema, en él se trabaja con los indicadores establecidos de forma individual.

10.5.2 Posible solución para ingresar un nuevo indicador a AS

De manera global, la automatización completa de la herramienta para el proceso de ingresar un nuevo indicador, es una tarea que en estos momentos puede llevarse a cabo, pero esto conlleva a una serie de configuraciones que constituyen de gran impacto en el AS y que necesitan de un determinado tiempo para poder realizarlos. Debido a las altas configuraciones que son necesarias para poder realizar de manera eficaz la labor de los indicadores, en el momento de la comunicación que tiene AS y ECLiPSe.

Sin embargo, hasta el momento se han probado ciertos cambios en la gráfica del programa, los cuales permitirían visualizar de manera gráfica, los cambios que debería tener AS para albergar nuevos indicadores. La idea, es seguir implementando la matriz base que utiliza el programa, para ir almacenando los distintos datos que fueron seleccionados por el usuario.

Para ingresar un nuevo indicador, los nombres se guardarán en un archivo .txt, el cual almacenara los nombres de los indicadores de acuerdo a la ilustración 52. De ésta manera es más fácil recorrer las líneas y tener límites para extraer el nombre del indicador.

El link que permitirá ingresar un nuevo indicador, es el que se muestra en la ilustración 53, éste se encuentra relacionado en la pestaña Configuración.

```
private static void calcular_indicadores(){
    indicador[actual][vf]    = VF();
    //indicador[actual][vfes]    = VFES();
    indicador[actual][tavfes]   = TAVFES();
    //indicador[actual][vfp]    = VFP();
    indicador[actual][tb]     = TB();
    indicador[actual][tavfps]   = TAVFPS();
    indicador[actual][tsb]   = TSB();
    indicador[actual][d_pr]   = d_pr();
    indicador[actual][d]   = d();
    indicador[actual][dmax_pr]   = dmax_pr();
    indicador[actual][dmax]   = dmax();
    indicador[actual][in1]   = In1();
    indicador[actual][in2]   = In2();
    //indicador[actual][in3]   = In3();
    indicador[actual][pvfes]   = PVFES();
    indicador[actual][ptavfes]   = PTAVFES();
    indicador[actual][pvfps]   = PVFPS();
    indicador[actual][ptavfps]   = PTAVFPS();
    indicador[actual][thrash]   = Thrash();
    indicador[actual][db] = DB();
    indicador[actual][vuDvtP] = VUDVTP();
```

Ilustración 50: Función para la actualización de indicadores compuestos.

```
%%
%%  Set_java_data
%%

set_java_data:-
    getval(strategy_used_g,Strategy_used_g),
    getval(node,NODE),
    getval(tv,TV),
    getval(vu,VU),
    getval(ss,SS),
    getval(sb,SB),
    getval(b,B),
        getval(val,VAL),
        getval(var,VAR),
        getval(in3,IN3),
        getval(vfes,VFES),
        getval(contadorvar,Contadorvar),

        setval(ssPorRest,1),
        getval(ssPorRest,SsPorRest), % Fitness SSrest/SStotal  a minimizar % Da error por que aveces es = Infinity Ahora solo

        write_exdr(eclipse_to_java,to_java(0,NODE,TV,VU,SB,B,Strategy_used_g,VAL,VAR,IN3,VFES,Contadorvar,0,SsPorRest)),
        flush(eclipse_to_java).
```

Ilustración 51: Código ECLiPSe para modificar datos de Java.

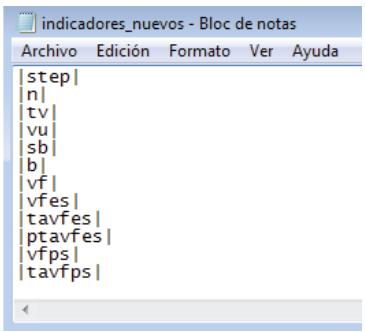


Ilustración 52: Forma de almacenamiento de los nombres de nuevos indicadores.

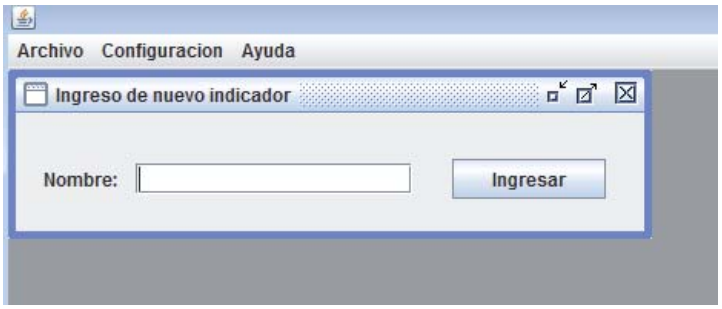


Ilustración 53: Ventana para ingresar un nuevo indicador.

10.5.2.1 Selección de indicadores.

Para la selección de indicadores, el usuario dispondrá de un combobox que desplegará todos los indicadores existentes en el programa. Para que el usuario pueda seleccionar el indicador, se debe seleccionar el estado que se encuentra al lado derecho del combobox, si el estado aparece marcado significa que el indicador que se visualiza en el combobox está seleccionado. La ilustración 54 muestra lo acontecido anteriormente. De igual manera, se establece para el Cutoff y el Fitness de la resolución.

Ilustración 54: Seleccionar un nuevo indicador.

10.5.2.2 Elección de Pesos mínimo, máximos y factores de suavizado.

Gráficamente es muy similar al proceso se selección de indicadores, salvo que éste posee la capacidad de poder establecer el valor del determinado peso o factor. La diferencia cae en los indicadores que mostrará el combobox, dado que sólo se desplegarán los seleccionados por el usuario. La Ilustración 55 muestra lo anterior.

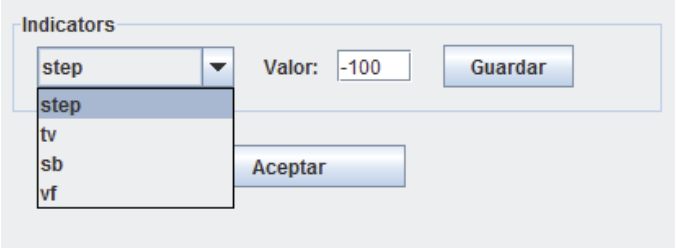


Ilustración 55: Seleccionar pesos mínimos de alfa.

10.5.3 Observaciones para ingresar un nuevo indicador de forma automática.

Antes de implementar el mecanismo para ingresar un nuevo indicador se debe considerar lo siguiente:

- Todas las configuraciones necesarias para que la comunicación entre AS y ECLiPSe se realice de forma correcta.
- La mayoría de las configuraciones, deberán analizar la existencia del indicador para poder ejecutar el código relacionado a él. De igual forma, se deberá verificar en el código implementado en el archivo del problema.

10.6 Estudio sobre el impacto del algoritmo genético en la resolución

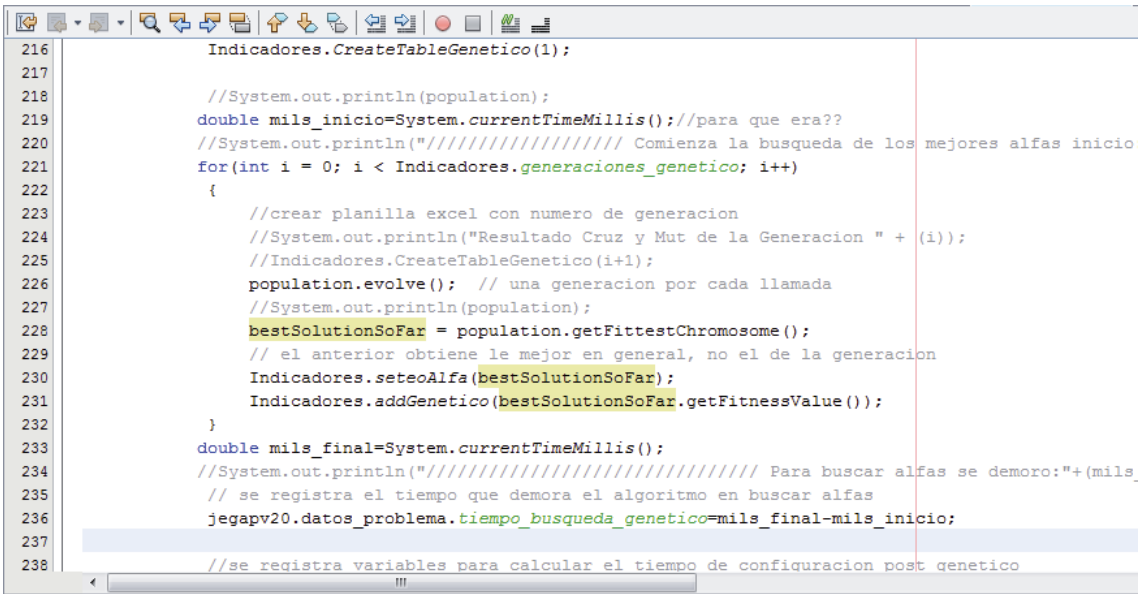
El algoritmo genético, hasta el momento, es el único optimizador que puede ser seleccionado para que participe en la resolución de un determinado problema. Por el momento, se poseen dos registros que tienen relación con el tiempo de participación, los cuales corresponden al tiempo total que demora la resolución del problema y el tiempo consumido por la herramienta ECLiPSe durante la ejecución de los steps. Pero éste último, no se muestra de forma directa en la hoja de resultados, sino que sólo se puede visualizar el comportamiento del porcentaje de tiempo consumido por los steps. Por lo tanto, la herramienta no entrega resultados en relación al tiempo de participación por parte del optimizador, tampoco muestra el tiempo consumido por ECLiPSe para la ejecución de los steps y el tiempo que se consume en ciertas configuraciones, el cual se denotará como overhead.

10.6.1 Cálculo del algoritmo genético y overhead

Para obtener el tiempo relacionado al algoritmo genético, se calcula el tiempo consumido por el bucle mostrado en la ilustración 56. Éste bucle, va obteniendo el mejor resultado de acuerdo al número de generaciones establecida.

Para obtener el tiempo relacionado al overhead, sólo se realiza la diferencia entre la suma del tiempo consumido por el algoritmo genético y eclipse en relación al tiempo total de la resolución del problema.

Finalmente para mostrar los resultados se confeccionó el cuadro de resumen de la ilustración 57, cuyas filas corresponde a distintas unidades de tiempo y las columnas corresponden al tiempo relacionado al algoritmo genético, overead y ECLiPSe.



```
216             Indicadores.CreateTableGenetico(1);
217
218             //System.out.println(population);
219            double mils_inicio=System.currentTimeMillis();//para que era??
220            //System.out.println("////////////////////// Comienza la busqueda de los mejores alfas inicio
221            for(int i = 0; i < Indicadores.generaciones_genetico; i++)
222             {
223                 //crear planilla excel con numero de generacion
224                 //System.out.println("Resultado Cruz y Mut de la Generacion " + (i));
225                 //Indicadores.CreateTableGenetico(i+1);
226                 population.evolve();  // una generacion por cada llamada
227                 //System.out.println(population);
228                 bestSolutionSoFar = population.getFittestChromosome();
229                 // el anterior obtiene le mejor en general, no el de la generacion
230                 Indicadores.seteoAlfa(bestSolutionSoFar);
231                 Indicadores.addGenetico(bestSolutionSoFar.getFitnessValue());
232             }
233            double mils_final=System.currentTimeMillis();
234            //System.out.println("////////////////////////////// Para buscar alfas se demoro:"+(mils_
235             // se registra el tiempo que demora el algoritmo en buscar alfas
236             jegapv20.datos_problema.tiempo_busqueda_genetico=mils_final-mils_inicio;
237
238             //se registra variables para calcular el tiempo de configuracion post genetico
```

Ilustración 56: Bucle correspondiente a la búsqueda que realiza el algoritmo genético.

	Genetic	Eclipse	Overhead	Total
Seconds	9,812	0,64861736	0,13138264	10,592
Mili	9812	648,61736	131,38264	10592
Nano	9812000000	648617360	131382640	10592000000

Ilustración 57: Cuadro resumen de los tiempos relacionados a la resolución de los problemas.

10.6.2 Impacto del algoritmo genético en la resolución

Para analizar el impacto que posee el algoritmo genético en una determinada resolución, se considera un análisis en base a 9 resultados obtenidos en milisegundos, de problemas relacionados y hasta ahora vistos en la herramienta como lo son N-Queen y Magic Square. La ilustración 58 muestra lo anterior.

De acuerdo a los datos registrados en la columna % Genetic, de la Ilustración 56, el promedio de participación que posee en la resolución el algoritmo genético es de aproximadamente el 90.87%.

10.7 Elección de choice function por medio de asistente

La manera actual de seleccionar los indicadores que conformarán la choice function es de tipo manual, es decir, el usuario marca los indicadores que van a participar y luego de esto, prosigue con la ejecución del programa. Para mejorar ésta manera de selección manual, se agregó un asistente que permite seleccionar, a través de un combobox, choice functions por defecto, las cuales se encuentran enumeradas y ordenadas de acuerdo a su ingreso. Esto permitirá que el usuario pueda elegir choice function con un sólo click y además se pueda tener información de las choice functions que se han utilizado anteriormente.

Problem	Genetic	Eclipse	Overhead	Total	% Genetic
n-queen(8)	1903	16,95565	249,0444	2169	87,74
n-queen(8)	1872	11,8946	207,1054	2091	89,53
n-queen(8)	1919	10,62146	223,3785	2153	89,13
n-queen(10)	2433	17,49168	233,5083	2684	90,65
n-queen(10)	2387	16,17245	232,8276	2636	90,55
n-queen(10)	2465	14,43815	235,5619	2715	90,79
n-queen(10)	2386	14,19936	236,8006	2637	90,48
magic square(4)	10842	380,7891	241,2109	11466	94,56
magic square(4)	10858	397,9803	241,0197	11497	94,44

Ilustración 58: Porcentaje de participación del algoritmo genético en la resolución de un problema.

10.7.1 Asistente de choice function

El asistente de choice function se encuentra conformado por un combobox y un checkbox. El combobox posee un listado de choice functions que han sido ingresadas, el usuario al momento de seleccionar el checkbox relacionado al asistente, estará aceptando que el programa utilice una choice function de su listado, eventualmente que ha seleccionado. En la ilustración 59 muestra el asistente de choice function.

10.7.2 Administración del asistente de choice function

Para ingresar una nueva choice function, se debe ingresar en la pestaña configuración del programa y luego seleccionar choice function (administrar). Én ésta sección, el usuario tiene la posibilidad de ingresar una nueva choice function nueva en el programa, ésta se realiza por medio de la selección manual de los indicadores que la conformarán. También el usuario, tiene la posibilidad de eliminar choice functions existentes. La ilustración 60 muestra lo anterior.

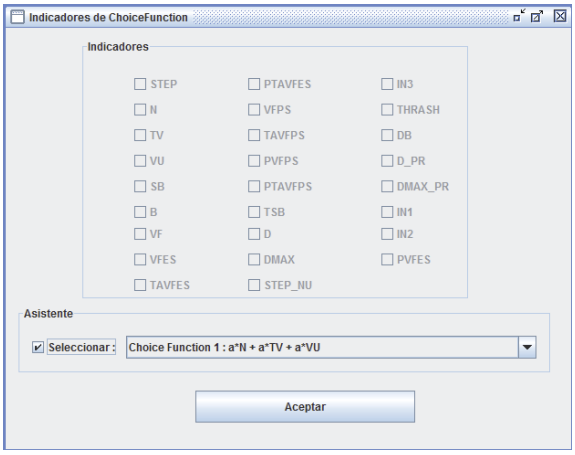


Ilustración 59: Asistente de Choice function.

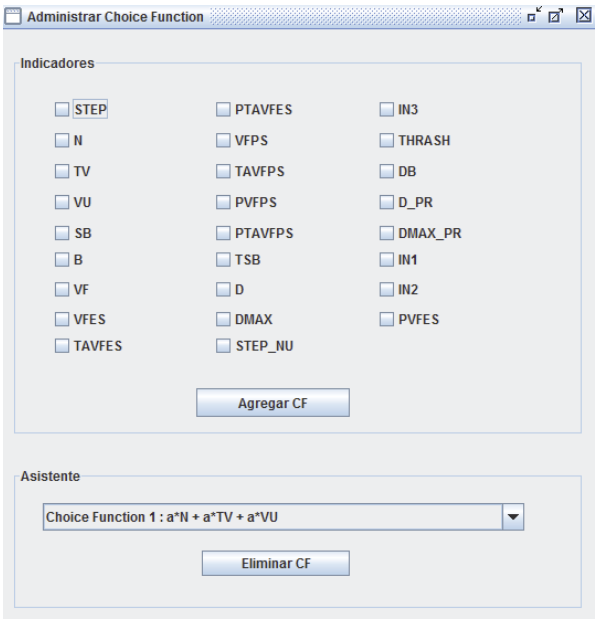


Ilustración 60: Administrar choice function.

Capítulo 11

Manual de Usuario

Ésta sección está destinada para que el usuario y los eventuales cambios que se realicen en la herramienta de manera gráfica, puedan verse de alguna manera reflejados como información para cualquier consulta y/o duda que pueda surgir, en el futuro, con respecto a algún funcionamiento del programa en sí. Cabe destacar, que posee variada información como por ejemplo: el funcionamiento correcto que se debe emplear para ejecutar el programa, distintos caminos que el usuario puede seleccionar durante los pasos de ejecución, información que el usuario debe entregar, cómo ingresar un nuevo problema, etc.

11.1 Detalles de la implementación

Para la implementación, se utilizó una librería opcional de Java llamada JavaHelp que permite colocar ventanas de ayuda, las cuales consisten en ficheros HTML (HyperText Markup Language), por lo tanto se pueden realizar tantos ficheros HTML como se requieran. También existe un fichero mapa de JavaHelp, éste consiste en un fichero XML (Extensible Markup Language) en el que se da una clave a cada uno de los ficheros HTML que se crean, por lo que cada fichero HTML posee un nombre, que servirá para identificarlo posteriormente. La Ilustración 61 muestra un fichero mapa basado en XML.

```
<?xml version='1.0' encoding='ISO-8859-1'?>
<!DOCTYPE map
PUBLIC "-//Sun Microsystems Inc.//DTD JavaHelp Map Version 2.0//EN"
"http://java.sun.com/products/javahelp/map_2_0.dtd">
<map version="1.0">
   <mapID target="aplicacion" url="html/main.html" />
   <mapID target="fitness" url="html/fitness.html" />
   <mapID target="cutoff" url="html/cutoff.html" />
   <mapID target="genetico" url="html/genetico.html" />
   <mapID target="ejecucion" url="html/ejecucion.html" />
   <mapID target="esquema" url="html/esquema.html" />
   <mapID target="ingresar_problema" url="html/ingresar_problema.html" />
   <mapID target="indicadores" url="html/indicadores.html" />
</map>
```

Ilustración 61: Fichero mapa de JavaHelp.

Una vez que el usuario selecciona “ayuda” desde el menú principal del programa, se muestra la imagen relacionada a la ilustración 62.

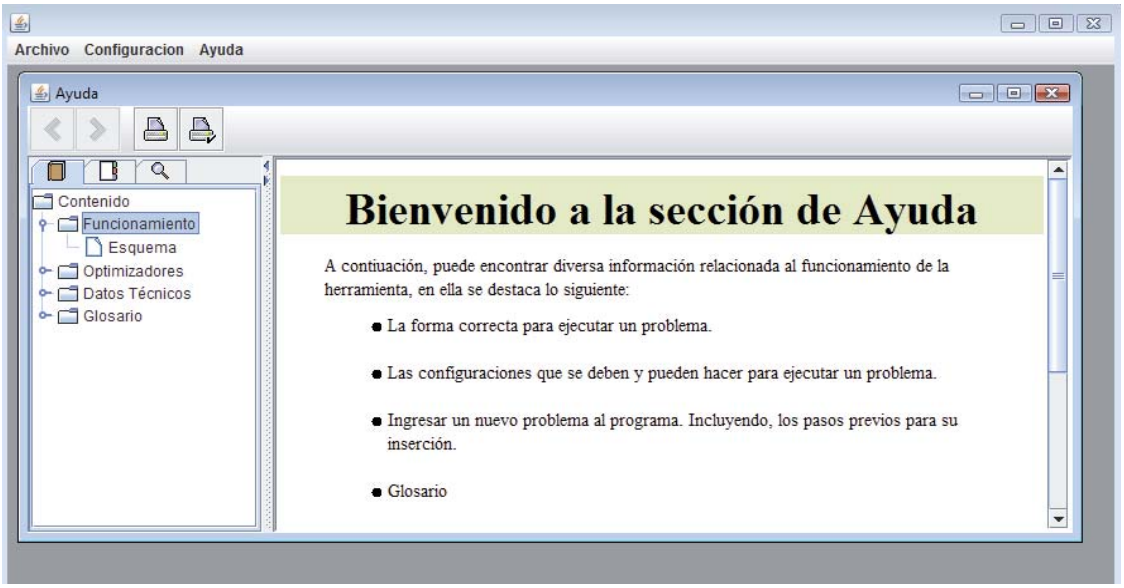


Ilustración 62: Página inicial del Manual de Usuario.

11.2 Sección: Funcionamiento

Ésta sección tiene como objetivo, describir cada uno de los pasos que el usuario tiene que seguir para realizar una correcta ejecución entorno a un problema. La Ilustración 63 muestra la sección relacionada a esto.

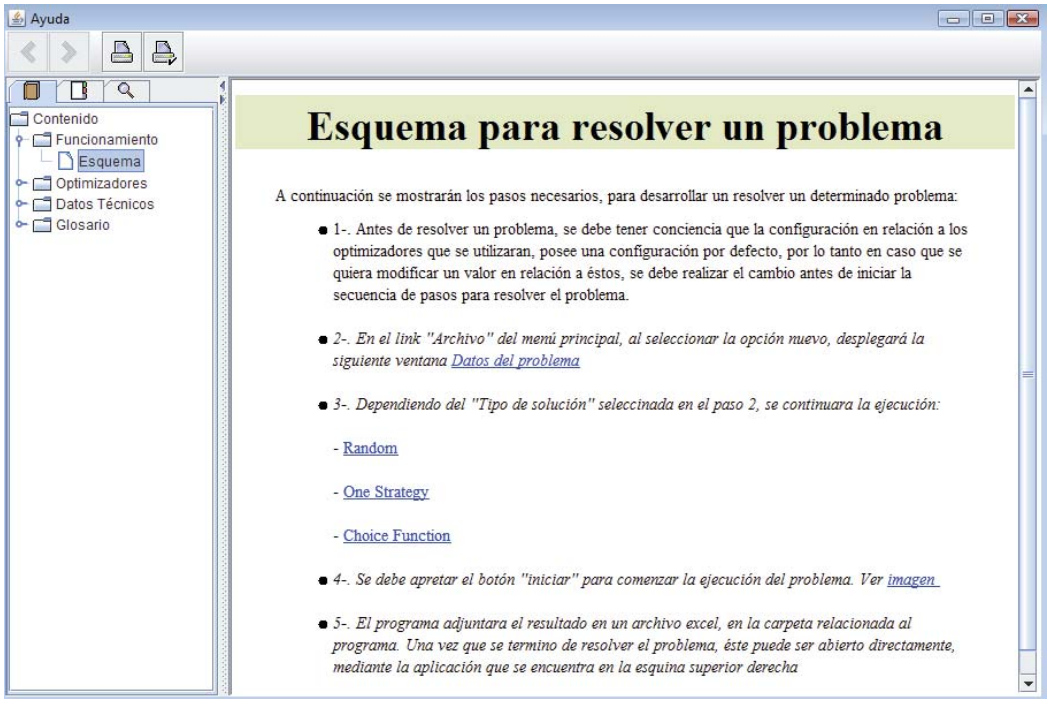


Ilustración 63: Sección Funcionamiento del manual de usuario.

11.3 Sección: Optimizadores

Sección destinada a las respectivas configuraciones que se deben realizar a los optimizadores que posee la herramienta. En éste caso, solamente existe el optimizador Algoritmo Genéticos, por lo tanto las configuraciones que se deben hacer, son respecto a él. En la ilustración 64 se muestras tres opciones que tiene el usuario, con respecto a la configuración del algoritmo genético. La configuración posee valores por defecto, por lo tanto, en caso que se requiera hacer un cambio extraordinario, se debe acudir a ésta parte del manual.

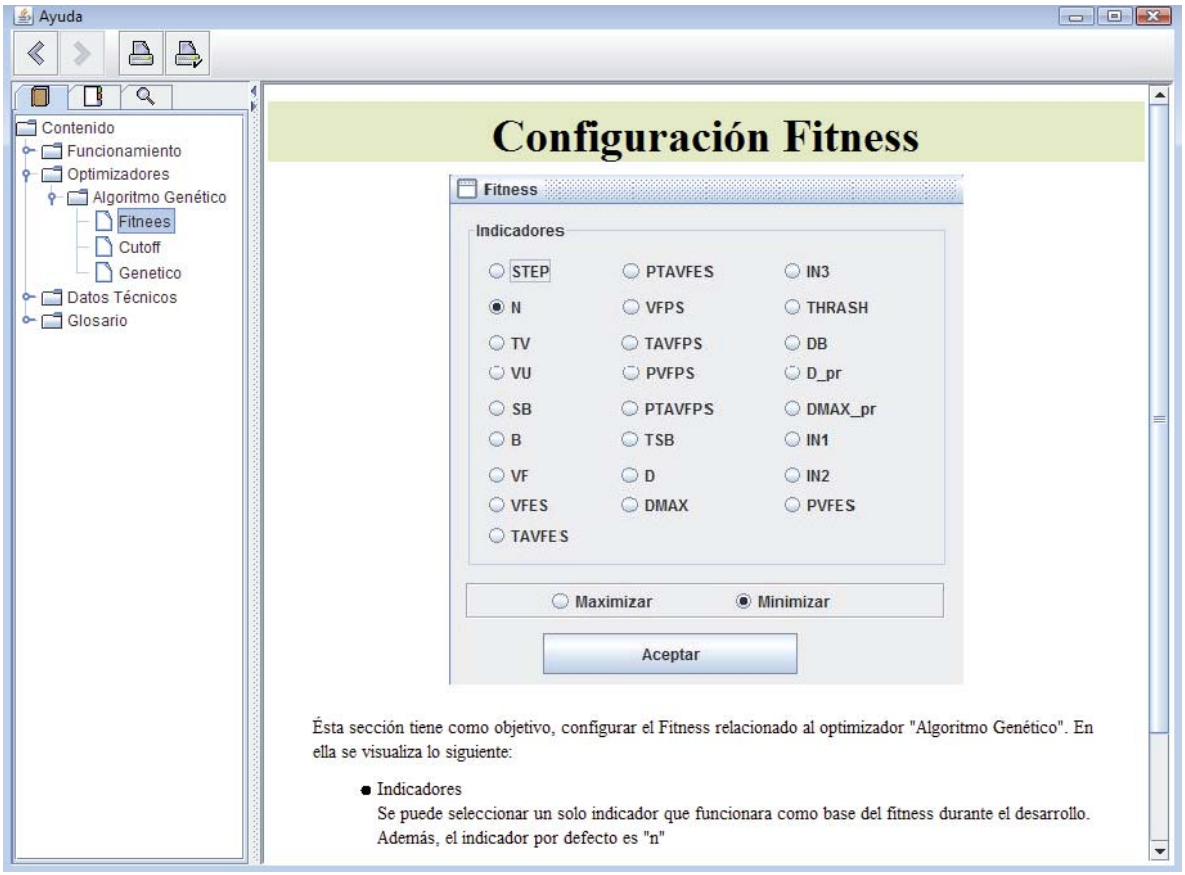


Ilustración 64: Configuración de optimizadores en el manual de usuario.

11.4 Sección: Datos Técnicos

Sección destinada para que el usuario sepa cuáles son los pasos necesarios para poder ingresar un nuevo problema al programa. El usuario tiene a su alcance las consideraciones generales que se deben tomar en cuenta, los pasos que se deben realizar en el programa para poder ingresar un nuevo problema y finalmente un ejemplo que detalla la manera en que se modela un problema nqueen, de tal forma, que se pueda resolver mediante la herramienta. La Ilustración 65 muestra lo anterior.

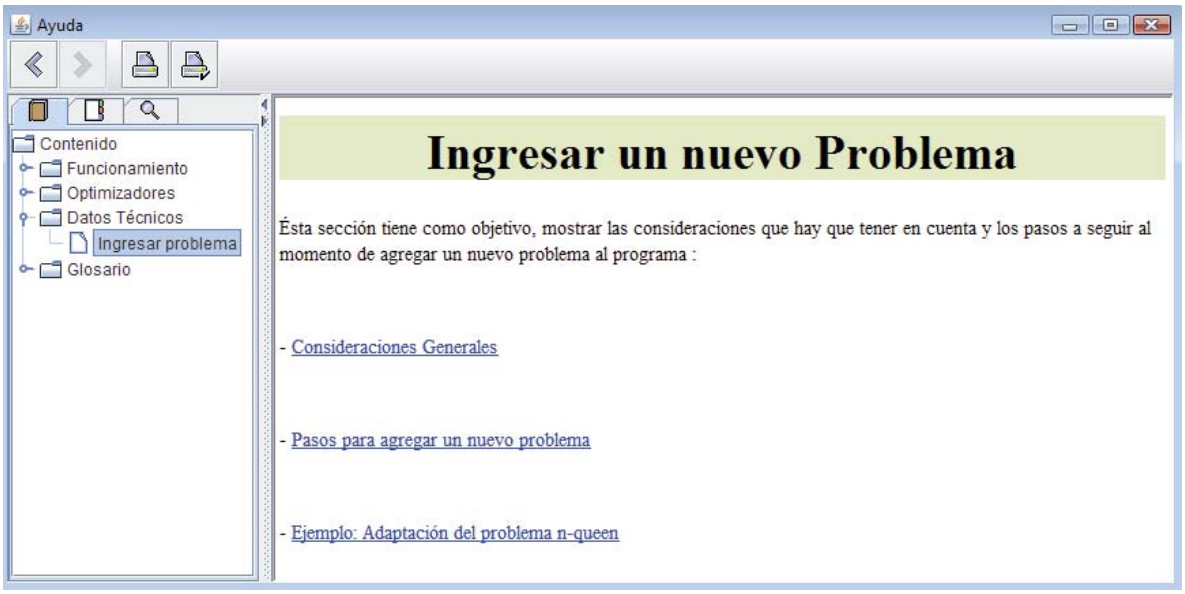


Ilustración 65: Datos para ingresar un nuevo problema mediante el manual de usuario.

11.5 Sección: Glosario

Ésta sección está destinada al significado de ciertas palabras que se utilizan en el programa. Uno de los principales objetivos es mostrar el significado que poseen algunos de los indicadores que se escogen durante la ejecución del programa. La Ilustración 66 muestra lo anterior.

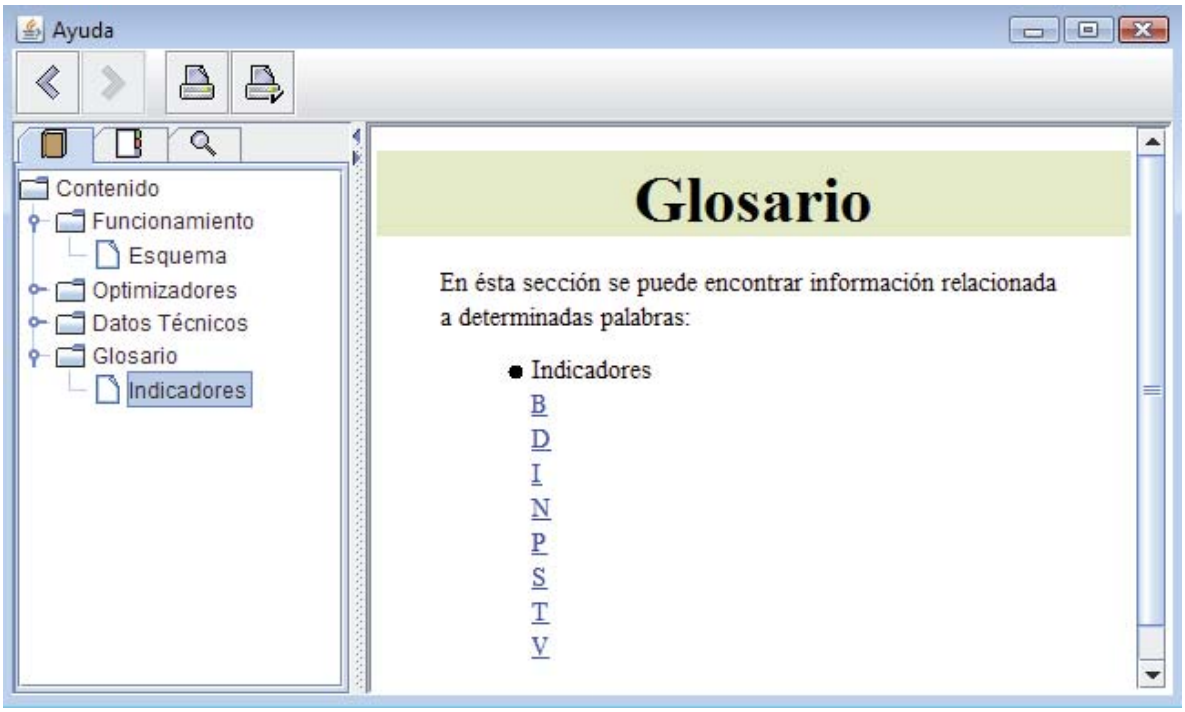


Ilustración 66: Glosario del manual de usuario.

Capítulo 13

Conclusiones

En el transcurso del desarrollo del proyecto, se ha abordado desde un principio que la utilización de un buen y adecuado modelado de problemas de satisfacción con restricciones, permiten obtener resultados exitosos y satisfactorios en el campo de resolución de problemas combinatorios complejos. Sin embargo, es importante señalar que se requiere de un alto conocimiento teórico para poder llevarla a cabo. Cabe destacar, los distintos métodos que hoy existen para poder desarrollar la tarea de resolución y a la vez considerar las distintas heurísticas utilizadas en la ejecución de éstas, puesto que es respetable tomar decisiones a partir de los distintos puntos de vista y resultados que generan ellas a través de su procesamiento.

Desde sus comienzos la Programación con Restricciones ha permitido a diversos investigadores modelar variados problemas bajo esta tecnología de software, indagando en diversas áreas como por ejemplo desde la investigación de operaciones hasta el área de biología molecular, para la secuenciación de DNA. A pesar de que estas áreas visualizan ser distintas, llevan consigo algo en común y es el tipo de problemas que se pueden generar en ellas, como en muchas otras.

La tendencia de los últimos años, de poder analizar el estado de progreso de resolución ha aumentado y esto con el fin de poder identificar automáticamente las buenas estrategias de desarrollo. Lo anterior es base para el proyecto, dado que se puede ver directamente en el framework que opera en el Autonomous Search, las búsquedas para hacer decisiones correctas de acuerdo a las estrategias de enumeración en el momento de la ejecución del problema.

Actualmente, existe una cantidad variada de lenguajes que soportan la Programación de Satisfacción de Restricciones, muchos de ellos basados en lenguajes previos como lo es Prolog. Algunos de ellos son B-Prolog, Mozart, ECLiPSe, GNU Prolog, etc. La actual implementación, de la cual se basará la futura arquitectura, está relacionada con el lenguaje ECLiPSe, debido a que se ha considerado un lenguaje eficiente en este tema, y se puede aprovechar la gran cantidad de información relacionada al uso de ECLiPSe.

El framework en el cual se basa Autonomous Search, apoya enormemente el objetivo que se pretende alcanzar, y los trabajos relacionados se ven enfocados a maximizar la eficiencia de él dando muy buenos resultados. Indicadores, Choice Functions y Snapshots se podrían considerar como grandes elementos constitutivos de la arquitectura actual. La información que poseen ellos, es crucial, para la toma de decisiones que realiza el Autonomous Search.

La herramienta desarrollada, fue de gran ayuda al momento de comenzar a desarrollar el proyecto, puesto que muchos de los puntos que no estaban del todo claro, pudieron esclarecerse al ver la funcionalidad de la herramienta, y de ésta manera ver cómo cada parte de la teoría lograba credibilidad. Ahora bien, si por un lado la herramienta fue de gran ayuda, por otro lado tuvo un cierto grado de complejidad en un comienzo, puesto que el programa obedecía un cierto orden y manera de programación, lo cual perjudicaba al realizar un posible cambio en él. Además, que el código es bastante extenso, por lo que se tenía que tener especial cuidado en su revisión.

Los resultados obtenidos, por medio de la función de selección resulta ser relativamente buena, puesto que a pesar de que quizás en algunos casos, no resulte ser la mejor solución, si logra tener un buen comportamiento frente a determinados problemas, en donde es primordial considerar, que en realidad se puede notar el buen comportamiento que la herramienta genera frente a varios problemas.

La compleja tarea de volver a rediseñar la interfaz gráfica, fue de gran ayuda, puesto que se pudo obtener ventajas y desventajas. Dentro de las ventajas, se adquirió un gran conocimiento de cada una de las funciones de la herramienta, se logra implementar nuevos mecanismos para la ayuda de pruebas, las cuales pueden resultar tediosas si se realizan individualmente. Dentro de las desventajas se pueden mencionar la alta cantidad de líneas de código, puesto que si bien se podían tratar algunas partes rápidamente, otras al estar tan dispersas las llamadas de funciones terminaban confundiendo el funcionamiento.

Una gran modificación al programa actual, es la incorporación de resultados mediante gráficos. Dado que si bien, tener el resultado numérico resulta importante, verlo visualmente permite obtener decisiones y conclusiones de manera más rápida, por lo tanto resulta un gran apoyo a la hora de comprender cuál fue por ejemplo, la heurística que más tiempo consumió durante el desarrollo.

La incorporación de dos nuevos problemas a Autonomous Search, permite poder acumular experiencia sobre la forma de adaptación que deben sufrir estos para poder integrarlos a la herramienta. Es importante reconocer si a primera vista se trata de problema de satisfacción de restricción o un problema de optimización de restricciones.

En relación al impacto que genera la búsqueda de alfas mediante el algoritmo genético, como optimizador, se pudo comprobar que la mayor parte del tiempo corresponde al algoritmo genético, se podría decir que por lo menos sobre el 85% corresponde a éste.

El ingreso automático de un nuevo indicador a Autonomous Search, en estos momentos se podría implementar pero considerando que este cambio conlleva modificar varias configuraciones en relación a éstos (puesto que no se pueden llegar y ejecutar sino que se deben comprobar de que estos existan), lo cual generaría gran impacto dentro del programa y también del tiempo destinado a esto.

En relación a la selección de choice function, ahora posee un asistente que permite escoger inmediatamente una de estas. Lo cual permitirá en un futuro que las choice

function se manejen en términos númericos y así relacionar de mejor forma cada una de ellas.

Una de las últimas implementaciones corresponde al manual de usuario. Éste es de gran ayuda, dado que esta destinado para todo aquel usuario que tenga intenciones de utilizar la herramienta. Se realizan descripciones en relación a los pasos que se deben seguir para ejecutar un problema, notas relacionadas con la configuración del optimizador, pasos a seguir y puntos a tener en cuenta al momento de ingresar un nuevo problema.

En relación a los trabajos futuros, resulta de especial interés analizar nuevos posibles optimizadores que se puedan incluir en la herramienta, para la eventual comparación entre cada uno de los respectivos resultados. Asimismo, la formulación de nuevos indicadores a partir de ciertos indicadores bases, sería una manera interesante de poder obtener aún mayor información con respecto a la ejecución de un problema.

Y finalmente, como fruto del proyecto realizado, es importante destacar que algunos de los resultados obtenidos de esta investigación, formaron parte del desarrollo relacionado a la publicación "*Using Autonomous Search for Generating Good Enumeration Strategy Blends in Constraint Programming"* [23]. La cual fue aceptada para ser presentada en la Conferencia Internacional sobre las Ciencias de la Computación y sus Aplicaciones (ICCSA 2012).

Bibliografía

[1] F. Barber and M.A. Salido: *Introduction to Constraint Programming*, Inteligencia Artificial, Revista Iberoamericana de Inteligencia Artificial, 20:13-30, 2003.

[2] J. C. Beck, P. Prosser, and R. J. Wallace. *Trying again to fail-first. In Workshop on Constraint Solving and Constraint Logic Programming (CSCLP),* volume 3419 of Lecture. Notes in Computer Science, pages 41-55. Springer, 2004.

[3] D. Frost, R. Dechter: *Look-ahead value orderings for constraint satisfaction problems*. In Proc. Of the Fourteenth International Joint Conference on Artificial Intelligence, 572-578, 1995.

[4] P. Sturdy. *Learning good variable orderings. In Proceedings of the 9th International Conference on Principales and Practice of Constraint Programming*, volume 2833 of Lecture Notes in Computer Science, page 997. Springer, 2003.

[5] M. Aranda Cabezas, *"Estrategias de Selección de Variables y Valores en la Resolución de Problemas Combinatoriales Utilizando Programación con Restricciones"*, in Informe Final Proyecto para Optar al título Profesional de Ingeniero Civil en Informática, P.U.C.V., 2007.

[6] E. Monfroy, C. Castro, B. Crawford. *Adaptative enumeration strategies and metabacktracks for constraint solving*. In Yakhno, TM., Neuhold, E. J., eds.: ADVIS. Volume 4243 of Lecture Notes in Computer Science., Springer (2006) 354-363.

[7] R. Bartak. *Constraint programming: In pursuit of the holy grail*. In Proceeding of WDS99 (invited lecture), Prague, June, 1999.

[8] B. Crawford, R. Soto, M. Montecinos, C. Castro and E. Monfroy. *A Hyperheuristic Approach for Dynamic Enumeration Strategy Selection in Constraint Satisfaction*. In proceedings of the 4th International Work-conference on the Interplay Between Natural and Artificial Computation (IWINAC), Volume Part II, pages 295-304, LNCS 6687, Springer, 2011.

[9] S. L. Epstein, E. C. Freuder, R. J. Wallace, A. Morozov, and B. Samuels. *The adaptive constraint engine. In Proceeding of the 8t International Conference on Principles and Practice of Contraint Programming (CP),* Lecture Notes in Computer Science, pages 525, 542. Springer, 2002.

[10] F. Boussemart, F. Hemery, C. Lecoutre, and L. Sais. *Boosting systematic search by weighting constraints*. In proceedings of the 16th European Conference on Artificial Intelligence (ECAI), pages 146-150. IOS Press, 2004.

[11] J. C. Beck, P. Prosser, and R. J. Wallace. *Variable ordering heuristics show promise. In Proceedings of the 10th International Conference on Principles and Practice of Constraint Programming (CP)*, volume 3258 of Lecture Notes in Computer Science, pages 711-715, 2004.

[12] R. M. Haralick and G. L. Elliot. *Increasing tree search efficiency for constraint satisfaction problems.* Artif. Intell, 14(3): 263-313, 1980.

[13] M. Berlinger *"Selector de Estrategias de Enumeración en Constraint Programming"*, in Informe Final Proyecto para Optar al título de Ingeniero Civil en Informática, P.U.C.V., 2010.

[14] M. Montecinos "*Parameter Setting for Adaptive Enumeration Strategies"* in Tesis de Grado Magister en Ingeniería Informática, P.U.C.V., 2008.

[15] F. Lorca *"Enfoque hiperheurístico en la resolución de problemas combinatorios"*. in Informe Final Proyecto para Optar al título de Ingeniero Civil en Informática, P.U.C.V., 2009.

[16] Y. Hamadi, E. Monfroy, and F. Saubion. *What is autonomous search?* Technical Report MSR-TR-2008-80. Microsoft Research, 2008.

[17] D. Santander *"Búsqueda autónoma de heurísticas de selección de variable y valor para la resolución de problemas de satisfacción de restricciones"* in Informe Final Proyecto para Optar al título de Ingeniero Civil en Informática, P.U.C.V., 2008.

[18] B. Crawford, R. Soto, M. Montecinos, C. Castro and Eric Monfroy "*A framework for Autonomous Search in the ECLiPSe Solver*". In proceedings of the 24th International Conference on Industrial Engineering and other Applications of Applied Intelligent Systems (IEA/AIE), Volume Part I, pages 79-84, LNCS 6703, Springer, 2011.

[19] B. Crawford "*Dynamic selection of enumeration strategies for solving constraint satisfaction problems*", in Thesis of the Degree of Doctor en Ingeniería Informática, U.T.F.S.M., 2011.

[20] C. Beck, P. Prosser, R, Wallace, "*Toward understanding variable ordering heuristics for constraint satisfaction problems*", in *Fourteenth Irish Artificial Intelligence and Cognitive Science Conference*, AICS, pp. 11-16,2003.

[21] S. L. Epstein, E. C. Freuder , R. J. Wallace, A. Morozov, B. Samuels, "*The adaptative Constraint Engine*". In P.V. Hentenryck, in CP Volume 2470 of Lecture Notes in Computer Science, Springer, pp. 525-542, 2002.

[22] S. Russell, R. Norvig, "*Artificial Intelligence : A Modern Approach*". Prentice Hall Pearson Education, 2003.

[23] R. Soto, B. Crawford, E. Monfroy, V. Bustos. *Using Autonomous Search for Generating Good Enumeration Strategy Blends in Constraint Programming*. In proceedings of the 12th International Conference on Computational Science and Its Applications (ICCSA), pages 607-617, volume 7335 of Lecture Notes in Computer Science, Springer, 2012.

Anexo

- Código fuente del problema NRP

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


nrp(Nurses, Paramedics,C_Nurses):-

  %%variables ------------------------------------

  %Nurses = 16,
  %Paramedics = 16,
  %C_Nurses = 4,
  Days = 28,

  dim(Schedule,[Nurses, Days]),
  Schedule[1..Nurses,1..Days] :: 0..2,

  dim(Schedule_Param,[Nurses, Days]),
  Schedule_Param[1..Paramedics,1..Days] :: 0..2,

  dim(Schedule_Chief,[C_Nurses, Days]),
  Schedule_Chief[1..C_Nurses,1..Days] :: 0..2,

  %%nurse constraints -----------------------------------
  %%-----------------------------------------------------

  %%constraint 1-2
  (for(I,1,Nurses),param(Days, Schedule) do
    (for(J,1,Days-1),param(I, Schedule) do
       Schedule[I,J]#=1 => Schedule[I,J+1]#=2,
       Schedule[I,J]#=2 => Schedule[I,J+1]#=0
    )
  ),

  %%constraint 3-4
  (for(I,1,Nurses),param(Days, Schedule) do
    (for(J,1,Days-2),param(I, Schedule) do
      (Schedule[I,J]#=0      and      Schedule[I,J+1]#=0)      =>
Schedule[I,J+2]#=1,
```

```
       Schedule[I,J]#=2       =>       (Schedule[I,J+1]#=0       and
Schedule[I,J+2]#=0)
    )
  ),

  %%occurrence constraints 5-7
  (for(J,1,Days),param(Schedule, Nurses) do
       Column is Schedule[1..Nurses,J],
       A is Nurses//2,
       occurrences(0,Column,A)
  ),
  (for(J,1,Days),param(Schedule, Nurses) do
       Column is Schedule[1..Nurses,J],
       B is Nurses//4,
       occurrences(1,Column,B)
  ),
  (for(J,1,Days),param(Schedule, Nurses) do
       Column is Schedule[1..Nurses,J],
       C is Nurses//4,
       occurrences(2,Column,C)
  ),

  %%preference constraints 8-13

  Schedule[3,1] #\= 0,
  Schedule[6,1] #= 1,Schedule[12,1] #= 1,
  Schedule[10,20] #= 0,Schedule[10,21] #= 0,
  Schedule[11,13] #= 0,Schedule[11,14] #= 0,
  Schedule[13,7] #= 0,Schedule[13,14] #= 0,
  Schedule[15,1] #= 0,Schedule[15,2] #= 0,


 %%paramedics constraints ------------------------------------
  %%-------------------------------------------------------
  %%constraint 1-2
  (for(I,1,Paramedics),param(Days, Schedule_Param) do
    (for(J,1,Days-1),param(I, Schedule_Param) do
       Schedule_Param[I,J]#=1 => Schedule_Param[I,J+1]#=2,
       Schedule_Param[I,J]#=2 => Schedule_Param[I,J+1]#=0
    )
  ),

  %%constraint 3-4
  (for(I,1,Paramedics),param(Days, Schedule_Param) do
    (for(J,1,Days-2),param(I, Schedule_Param) do
      (Schedule_Param[I,J]#=0   and   Schedule_Param[I,J+1]#=0)   =>
Schedule_Param[I,J+2]#=1,
       Schedule_Param[I,J]#=2   =>   (Schedule_Param[I,J+1]#=0   and
Schedule_Param[I,J+2]#=0)
    )
  ),
```

```
  %%occurrence constraints 5-7
  (for(J,1,Days),param(Schedule_Param, Paramedics) do
       Column is Schedule_Param[1..Paramedics,J],
       A is Paramedics//2,
       occurrences(0,Column,A)
  ),
  (for(J,1,Days),param(Schedule_Param, Paramedics) do
       Column is Schedule_Param[1..Paramedics,J],
       B is Paramedics//4,
       occurrences(1,Column,B)
  ),
  (for(J,1,Days),param(Schedule_Param, Paramedics) do
       Column is Schedule_Param[1..Paramedics,J],
       C is Paramedics//4,
       occurrences(2,Column,C)
  ),

  %%preference constraints 1-4

  Schedule_Param[3,1] #\= 0,
  Schedule_Param[6,20] #= 0,Schedule_Param[6,21] #= 0,
  Schedule_Param[8,13] #= 0,Schedule_Param[8,14] #= 0,
  Schedule_Param[11,7] #= 0,Schedule_Param[11,14] #= 0,

  %%chief nurse constraints 1-5------------------------------
  %%-------------------------------------------------------

  (for(J,1,Days),param(Schedule,Schedule_Chief,    Nurses,C_Nurses)
do
       Seq is Schedule[1..Nurses,J],
       Seq_C is Schedule_Chief[1..C_Nurses,J],
       occurrences(1,Seq,N),
       occurrences(2,Seq,D),
       occurrences(2,Seq_C,S),
       (N<D) => (S#>=D-N)
  ),

  (for(J,1,Days),param(Schedule,Schedule_Chief,    Nurses,C_Nurses)
do
       Seq is Schedule[1..Nurses,J],
       Seq_C is Schedule_Chief[1..C_Nurses,J],
       occurrences(1,Seq,N),
       occurrences(2,Seq,D),
       occurrences(1,Seq_C,S),
       (N>D) => (S#>=N-D)
  ),

  % 2 nurses per day  or 2 nurses per night
  (for(J,1,Days),param(Schedule_Chief, C_Nurses) do
       Seq_C is Schedule_Chief[1..C_Nurses,J],
       occurrences(1,Seq_C,D),
       occurrences(2,Seq_C,N),
```

```
      DN is C_Nurses//2,
     % NN is C_Nurses//4,

      ((D #= DN) or (N #= DN))% or ((N #>= NN) and (D #>= NN)))
  ),

  % no more than 3 consecutive day shifts
  (for(I,1,C_Nurses),param(Days, Schedule_Chief) do
    (for(J,1,Days-3),param(I, Schedule_Chief) do
       (Schedule_Chief[I,J]#=1 and
       Schedule_Chief[I,J+1]#=1 and
       Schedule_Chief[I,J+2]#=1)  => Schedule_Chief[I,J+3]#=0
    )
  ),

  %% constraint 5
  (for(I,1,C_Nurses),param(Schedule_Chief) do
       Seq_1 is Schedule_Chief[I,1..7],
       Seq_2 is Schedule_Chief[I,8..14],
       Seq_3 is Schedule_Chief[I,15..21],
       Seq_4 is Schedule_Chief[I,22..28],
       occurrences(1,Seq_1,A),
       occurrences(2,Seq_1,B),
       occurrences(0,Seq_1,C),
       (((A #= 4) or (B #= 3)) and (C#=2)),

       occurrences(1,Seq_2,C),
       occurrences(2,Seq_2,D),
       occurrences(0,Seq_2,C),
       (((C #= 4) or (D #= 3)) and (C#=2)),

       occurrences(1,Seq_3,E),
       occurrences(2,Seq_3,F),
       occurrences(0,Seq_3,C),
       (((E #= 4) or (F #= 3)) and (C#=2)),

       occurrences(1,Seq_4,H),
       occurrences(2,Seq_4,L),     occurrences(0,Seq_4,C),
       (((H #= 4) or (L #= 3)) and (C#=2))
  ),

initialize([], Schedule, List_A),
initialize(List_A, Schedule_Param, List_B),
initialize(List_B, Schedule_Chief, List_C),

solve(List_C,Var,Val),

print_board1(Schedule, Nurses, Days),
print_board2(Schedule_Param, Paramedics, Days),
print_board3(Schedule_Chief, C_Nurses, Days).
```

```
%--------------------------------------------
%Round_Robin(from left to right) + Value ordering(min,max,middle)

solve(List,rr,Value_Ord):-
   ( foreach(Var,List),param(Value_Ord) do indomain(Var,Value_Ord)
   ).

%--------------------------------------------
%Middle_out variable ordering + Value ordering(min,max,middle)

solve(List,mo,Value_Ord):-
   middle_out(List,MOutList),
   solve(MOutList,ff,Value_Ord).

%--------------------------------------------
%First_fail variable ordering + Value ordering(min,max,middle)

solve(List,ff,Value_Ord):-
   ( fromto(List, Vars, Rest,[]),param(Value_Ord)
     do
     delete(Var,Vars,Rest,0,first_fail),
     indomain(Var,Value_Ord)
   ).

%--------------------------------------------
%anti_first_fail       variable       ordering       +       Value
ordering(min,max,middle)

solve(List,aff,Value_Ord):-
   ( fromto(List, Vars, Rest,[]),param(Value_Ord)
     do
     delete(Var,Vars,Rest,0,anti_first_fail),
     indomain(Var,Value_Ord)
   ).

middle_out(List,MOutList) :-
  halve(List,FirstHalf,LastHalf),
  reverse(FirstHalf, RevFirstHalf),
  splice(LastHalf,RevFirstHalf,MOutList).

%%-----------------------------------------------------

print_board1(Schedule, Nurses, Days):-
    dim(Schedule,[Nurses, Days]),
      (for(I,1,Nurses), param(Schedule, Days) do
        (for(J,1,Days), param(Schedule, I) do
          X is Schedule[I,J],
          printf(" %d",[X])
        ),nl
       ),nl.

print_board2(Schedule_Param, Paramedics, Days):-
```

```
    dim(Schedule_Param,[Paramedics, Days]),
      (for(I,1,Paramedics), param(Schedule_Param, Days) do
        (for(J,1,Days), param(Schedule_Param, I) do
          X is Schedule_Param[I,J],
          printf(" %d",[X])
        ),nl
       ),nl.

print_board3(Schedule_Chief, C_Nurses, Days):-
    dim(Schedule_Chief,[C_Nurses, Days]),
      (for(I,1,C_Nurses), param(Schedule_Chief, Days) do
        (for(J,1,Days), param(Schedule_Chief, I) do
          X is Schedule_Chief[I,J],
          printf(" %d",[X])
        ),nl
       ),nl.

% -----------------------------------------------------------------
% initializes variables of the model
initialize(PrevList,             CurrList,             NextList):-
flatten_vector(CurrList,RevFlatList),append(PrevList,RevFlatList,N
extList).

% returns a flat list representing from an array or matrix
flatten_vector(CurrList,RevFlatList):-term_variables(CurrList,
FlatList),rev(FlatList,RevFlatList).

% reverses a list
rev(L,R) :- accRev(L,[],R).
accRev([H|T],A,R) :- accRev(T,[H|A],R).
accRev([],A,A).
```

- Código fuente del problema MCDP

```
:-lib(ic).
:-lib(ic_global).
:-lib(ic_search).
:-lib(branch_and_bound).
:-lib(lists).
:-lib(ic_search).
:-lib(listut).

go(Cells,Board,Vars) :-

        % The problem instance:
     Matrix_Machine_Part = [](1, 0, 0, 0, 1, 1, 1),
                            [](0, 1, 1, 1, 1, 0, 0),
                            [](0, 0, 1, 1, 1, 1, 0),
                            [](1, 1, 1, 1, 0, 0, 0),
                            [](0, 1, 0, 1, 1, 1, 0)),
```

```
        %Cells = 3,
        Mmax = 4,
        Sum = 1,

        % TODO: We should probably do the output here...
        mcdp(Matrix_Machine_Part,      Cells,      Mmax,      Sum,
_Matrix_Machine_Cell, _Matrix_Part_Cell,Board,Vars).

mcdp(Matrix_Machine_Part,  Cells,  Mmax,  Sum,  Matrix_Machine_Cell,
Matrix_Part_Cell,Board,Vars) :-

        dim(Matrix_Machine_Part,[Machines, Parts]),

        % Machines = 5,
        % Cells = 3,
        % Parts = 7,
        % Mmax is 4,
        % Sum = 1,

        writeln([machines:Machines, cells:Cells, parts:Parts,
                 mmax:Mmax, sum:Sum]),

        %
        % Machines assignment
        %
        dim(Matrix_Machine_Cell, [Machines,Cells]),             %
make the Machine and Cell variables
     Matrix_Machine_Cell[1..Machines, 1..Cells] :: 0..1,   %    set
the domains

        % Check for maximum number of machines in each cell
     ( for(J,1,Cells),
         param(Machines,Matrix_Machine_Cell,Mmax) do
             sum(Matrix_Machine_Cell[1..Machines,J]) #=<  Mmax
     ),

        % Ensure uniqueness of assignment
        ( for(I,1,Machines),
         param(Cells,Matrix_Machine_Cell,Sum) do
             sum(Matrix_Machine_Cell[I,1..Cells]) #=  Sum
     ),

        % Dual model
        dim(AssignedMachines, [Machines]),
        AssignedMachines :: 0..Cells,

        ( for(I,1,Machines) * for(K,1,Cells),
         param(Matrix_Machine_Cell, AssignedMachines) do
             (AssignedMachines[I]         #=         K)         #=
(Matrix_Machine_Cell[I,K] #= 1)
        ),
```

```
        %
        % Parts assignments
        %
        % make the Parts and Cell variables
      dim(Matrix_Part_Cell, [Parts,Cells]),
      Matrix_Part_Cell[1..Parts, 1..Cells] :: 0..1,

        % Ensure uniqueness of assignment
      ( for(I,1,Parts),param(Cells,Matrix_Part_Cell, Sum) do
           sum(Matrix_Part_Cell[I,1..Cells]) #=  Sum
      ),

        % Dual model
        dim(AssignedParts, [Parts]),
        AssignedParts :: 0..Cells,

        ( for(J,1,Parts) * for(K,1,Cells),
          param(Matrix_Part_Cell, AssignedParts) do
              (AssignedParts[J] #= K) #= (Matrix_Part_Cell[J,K] #=
1)
        ),

        %
        % Calculate the TotalCost
      %
        MP is Machines*Parts,
        TotalCost :: 0..MP,
      ( for(K,1,Cells) * for(I,1,Machines)* for(J,1,Parts),
          fromto(0,In,Out,TotalCost),

param(Matrix_Machine_Part,Matrix_Machine_Cell,Matrix_Part_Cell) do
                A is Matrix_Machine_Part[I,J],
                Z is Matrix_Part_Cell[J,K],
                Y is Matrix_Machine_Cell[I,K],
                Out #= In + A*Z*(1-Y)
      ),
        %
        % Dual version of TotalCost
        %
        TotalCost2 :: 0..MP,
      ( for(K,1,Cells) * for(I,1,Machines)* for(J,1,Parts),
          fromto(0,In,Out,TotalCost2),

param(Matrix_Machine_Part,AssignedMachines,AssignedParts) do
                A is Matrix_Machine_Part[I,J],
                Out #= In + (A #= 1 and
                                AssignedMachines[I] #= K and
                                AssignedParts[J] #\= K)
      ),

        % hakank: symmetry breaking,
        % assign Machine 1 to Cell 1
```

```
      % Matrix_Machine_Cell[1,1] #= 1,

         % assign Part 1 to Cell 1
      % Matrix_Part_Cell[1,1] #= 1,

         %
         % search
term_variables([Matrix_Machine_Cell,Matrix_Part_Cell],Vars),
      % term_variables([AssignedParts, AssignedMachines],Vars),
      Board = [AssignedMachines,AssignedParts],
      term_variables(Board,Vars),
      %
term_variables([Matrix_Machine_Cell,Matrix_Part_Cell,AssignedMachi
nes,AssignedParts],Vars),

         % orig:
      %  minimize(search(Vars,  0,  anti_first_fail,  indomain_min,
complete,[backtrack(BT)]),TotalCost),

         % hakank: testing
         % TotalCost
      % minimize(search(Vars, 0, anti_first_fail, indomain_median,
complete,[backtrack(BT)]),TotalCost),
      %   minimize(search(Vars,   0,   occurrence,   indomain_min,
complete,[backtrack(BT)]),TotalCost),
      %  minimize(search(Vars,  0,  max_regret,  indomain_median,
complete,[backtrack(BT)]),TotalCost),

         % TotalCost2
         minimize(search(Vars,  0,  most_constrained,  indomain_min,
complete,[backtrack(BT)]),TotalCost2),
         %   minimize(search(Vars,   0,   max_regret,   indomain_min,
complete,[backtrack(BT)]),TotalCost2),

         %   minimize(search(Vars,   0,   first_fail,   indomain_min,
complete,[backtrack(BT)]),TotalCost2),

      writeln(total_cost:TotalCost),
      writeln(total_cost2:TotalCost2),
      writeln(backtracks:BT),

         % hakank: nice printout
         writeln('Machines:'),
         writeln(assigned_machines:AssignedMachines),
         % print_matrix(Matrix_Machine_Cell),
         nl,
         writeln('Parts:'),
         writeln(assigned_parts:AssignedParts),
         % print_matrix(Matrix_Part_Cell),
         nl.

% hakank: print result
```

```
print_matrix(X) :-
        dim(X,[M,_N]),
        ( for(J,1,M),
          foreacharg(Row, X) do
              collection_to_list(Row,List),
              printf("%3d: ", J),
              ( foreach(L,List) do
                    printf("%3d", L)
              ),
              nl
        ).
```