Hiperheurísticas Mediante un Enfoque Neuro-Evolutivo para el Ordenamiento Dinámico de Variables en Problemas de Satisfacción de Restricciones

T E S I S

Maestría en Ciencias en Sistemas Inteligentes

Instituto Tecnológico y de Estudios Superiores de Monterrey

Por

LCC. Jorge Iván Fuentes Rosado

Diciembre 2008

Hiperheurísticas Mediante un Enfoque Neuro-Evolutivo para el Ordenamiento Dinámico de Variables en Problemas de Satisfacción de Restricciones

TESIS

Maestría en Ciencias en

Sistemas Inteligentes

Instituto Tecnológico y de Estudios Superiores de Monterrey

Por

LCC. Jorge Iván Fuentes Rosado

Diciembre 2008

Instituto Tecnológico y de Estudios Superiores de Monterrey

División de Graduados en Mecatrónica y Tecnologías de Información

Los miembros del comité de tesis recomendamos que la presente tesis de Jorge Iván Fuentes Rosado sea aceptada como requisito parcial para obtener el grado académico de Maestro en Ciencias en:

Sistemas Inteligentes

Comité de tesis:

Dr. Hugo Terashima Marín
Asesor de la tesis

Dr. Manuel Valenzuela Rendón
Sinodal

Dr. Santiago E. Conant Pablos
Sinodal

Dr. Joaquín Acevedo M.
Director de Investigación y Posgrado
Escuela de Ingeniería

Diciembre de 2008

Hiperheurísticas Mediante un Enfoque Neuro-Evolutivo para el Ordenamiento Dinámico de Variables en Problemas de Satisfacción de Restricciones

Por

LCC. Jorge Iván Fuentes Rosado

TESIS

Presentada a la División de Mecatrónica y Tecnologías de Información

Este trabajo es requisito parcial para obtener el grado académico de Maestro en Ciencias en Sistemas Inteligentes

Instituto Tecnológico y de Estudios Superiores de Monterrey

Campus Monterrey

Monterrey, N.L. Diciembre de 2008

A todos aquellos que luchan por alcanzar sus sueños.

Reconocimientos

A mi asesor el Dr. Hugo Terashima Marín por su apoyo y dirección durante la elaboración de esta tesis.

A mis sinodales el Dr. Manuel Valenzuela y el Dr. Santiago Conant por el tiempo que le dedicaron a la revisión de esta tesis.

A mi madre, Sra. Sara Esther Rosado Díaz por la confianza que tiene en mí y alentarme a seguir mis sueños.

A mi padre, Sr. Jorge Alber Fuentes Ávila por quererme y apoyarme en las decisiones que tomo.

A mis hnos. Pepe y Sugeidy por apoyarme en todas las decisiones de mi vida.

A Doris por toda la ayuda prestada y por escucharme cuando lo necesité.

A todos mis profesores por compartir sus conocimientos dentro y fuera de las aulas de clase.

A mis compañeros de clase y amigos de la maestría por los momentos que compartimos.

A todos mis nuevos amigos que hicieron de mi estancia en Monterrey una de las experiencias más gratificantes de mi vida. Los llevaré por siempre en la mente y corazón.

A todos aquellos que desde Yucatán me apoyaron y estuvieron pendientes de mí en todo momento. En especial a Licho porque siempre estuvo allí cuando lo necesité y porque sé que siempre estará. Muchas Gracias Bro.

A mis compañeros y amigos del Harmon Hall Centro por todo lo que aprendí de ustedes.

Jorge Iván Fuentes Rosado

Instituto Tecnológico y de Estudios Superiores de Monterrey
Diciembre 2008

Hiperheurísticas Mediante un Enfoque Neuro-Evolutivo para el Ordenamiento Dinámico de Variables en Problemas de Satisfacción de Restricciones

Jorge Iván Fuentes Rosado, M.C.
Instituto Tecnológico y de Estudios Superiores de Monterrey, 2008

Asesor de la tesis: Dr. Hugo Terashima Marín

Muchos de los problemas de las Ciencias de la Computación y de la Inteligencia Artificial (IA) pueden ser caracterizados como problemas de Satisfacción de Restricciones. Un CSP, por sus siglas en inglés, está compuesto por un conjunto de variables, un conjunto dominio de posibles valores y una serie de restricciones entre variables a ser satisfechas. Encontrar una solución para un problema de satisfacción de restricciones consiste en encontrar una asignación consistente para todo el conjunto de variables de tal manera que satisfaga todas las restricciones. Los algoritmos de solución más conocidos son *Backtracking*, *Foward Checking* y otros híbridos los cuales están basados en la teoría de algoritmos de búsqueda en árboles. La solución de estos problemas cuando son considerados difíciles por métodos tradicionales se vuelve muy lento y tedioso puesto que el espacio de búsqueda es muy grande, es por ello que se han realizado muchas investigaciones para la optimización de la forma de solucionar estos problemas. Además, existen instancias de CSPs que son particularmente difíciles de resolver, ya que requieren una elevada cantidad de verificaciones de consistencia para encontrar una solución o demostrar que no existe alguna. El ordenamiento de las variables juega un papel de suma importancia en la búsqueda de la solución. Un buen ordenamiento de variables al momento de buscar una solución representa una menor complejidad en la búsqueda. Existen dos enfoques para el ordenamiento de variables. El primer enfoque es conocido como en ordenamiento estático el cual consiste en definir el orden en la cual las variables estarán instanciadas antes de empezar a solucionar el problema.El otro enfoque es conocido como ordenamiento dinámico el cual consiste en determinar la variable a instanciar en tiempo de solución. Para esto surgieron algunas heurísticas que ayudan en este proceso, de las cuales podemos nombrar *Fail First*, *Rho*, *Kappa*, entre

otras, pero ninguna de ellas ha demostrado ser eficiente para todas las instancias. En este enfoque la heurística elegida es aplicada en todos los niveles del árbol sin importar las modificaciones del problema. Recientemente se ha empezado a trabajar con un enfoque basado en hiperheurísticas, el cual, dada las características del estado del problema determina que heurística debe ser aplicada para elegir la variable a ser asignada, en éste modelo en cada nivel del árbol la heurística aplicada puede variar respecto a la anterior. Este modelo se compone de una etapa de entrenamiento y una de prueba. La fase de entrenamiento consiste en utilizar un algoritmo genético para evolucionar una población de redes neuronales. Las redes neuronales son entrenadas por medio del algoritmo de retropropagación del error y después evaluadas con diferentes instancias de CSPs. La red neuronal tiene como entrada el estado del problema y como salida la heurística sencilla a aplicar. La intención de utilizar diferentes instancias de CSPs durante la etapa de entrenamiento es desarrollar procesos de solución generales que puedan ser aplicados para resolver un gran número de instancias, más que encontrar buenas soluciones para instancias específicas. La idea general es la siguiente, dado el estado del CSP P es utilizado para alimentar la red neuronal de la cual se obtendrá como salida la heurística sencilla a aplicar. Una vez aplicada la heurística el CSP P cambia a un CSP P', el cual es una version actualizada del anterior. El procedimiento se repite hasta resolver el problema completamente. Al terminar el entrenamiento de la población se toma el mejor individuo y es utilizado para solucionar tanto los problemas utilizados para su entrenamiento como problemas nuevos. Lo anterior para hacer comparaciones con los resultados de la aplicación individual de las heurísticas simples. Los resultados demuestran que es factible la creación de una hiperheurística bajo este modelo de solución, puesto que las hiperheurísticas generadas se comportaron como la mejor heurística sencilla a aplicada y con esto mejor que el promedio que ellas. La mejor heurística simple puede ser diferente en cada instancia del problema.

Índice general

Índice de cuadros

Índice de figuras

Capítulo 1

Introducción

Muchos de los problemas del mundo real, debido a su complejidad y tamaño, no son posibles de resolver de manera exacta, es por ello que el campo de la Inteligencia Artificial (IA), se caracteriza por el uso de algoritmos de aproximación, los cuales son más rápidos que los algoritmos exactos y tan confiables como estos últimos.

Los *Problemas de Satisfacción de Restricciones* (CSPs, por sus siglas en inglés, *Constraint Satisfaction Problems*) cuentan con características muy establecidas, y muchos problemas que se presentan en IA pueden ser mapeados dentro de este tipo. Las áreas de visión, programación de tareas, planeación, razonamiento temporal, diseños de planes de piso, diseños de circuitos, etc. pueden ser resueltos como CSPs [20]. Todo tipo de avance en la resolución de este tipo de problemas beneficiaría a todas las áreas antes mencionadas. Un CSP está definido por un conjunto finito de variables, un dominio finito de posibles valores para las variables y un conjunto de restricciones entre variables, las cuales describen las posibles combinaciones variable/valor válidas [9]. La solución de un CSP es un conjunto de asignaciones consistentes, es decir, que todas las variables tengan asignadas un valor y éstas satisfagan todas las restricciones. Algunos CSPs requieren soluciones que maximicen su función objetivo [14].

Al momento de realizar un algoritmo general para la solución de CSPs se tiene que tomar en cuenta lo siguiente: *a)* Qué variable será asignada en el siguiente paso y qué valor se evaluará primero, *b)* Qué implicaciones tiene la asignación actual sobre las variables no asignadas y *c)* Qué hacer cuando la ruta falla, es decir, cuando no se encuentra una solución [14]. Existen algoritmos que se encargan de solucionar las cuestiones *b* y *c*, como pueden ser el *Backtracking*, *Foward Checking*, *Arc Consistency*, entre otros, de los cuales se hablará en las secciones subsecuentes. Para el primer inciso, el orden de las variables de acuerdo con varios investigadores tiene un fuerte impacto en la solución de los CSPs. Para el ordenamiento de variables existen dos enfoques: el estático, donde el orden se establece antes de empezar la solución y se mantiene hasta el final; y el dinámico, el orden de las variables se va modificando a lo largo de la solución. Experimentos realizados han demostrado que un orden de tipo dinámico ofrece una mejora significativa en la eficiencia del proceso de búsqueda que en aquella obtenida por un orden de tipo estático [9].

Tiempo atrás el problema de ordenamiento dinámico se ha resuelto a través de heurísticas simples, que son reglas que permiten orientar a un algoritmo hacia la solución. Dado un problema se aplica la misma heurística hasta encontrar la solución. Algunas de estas heurísticas más costosas computacionalmente hablando que otras. Investigaciones recientes en otros ámbitos proponen la utilización de hiperheurísticas, que son reglas que permiten elegir dentro de un conjunto de heurísticas, la heurística que mejor se aplique al estado actual del problema. En el trabajo de Terashima et al. [21] se hace uso de hiperheurísticas para el corte de material, y en otro trabajo de Terashima et al. [9] del uso de los algoritmos genéticos para el ordenamiento dinámico de las variables en los problemas de satisfacción de restricciones y recientemente en un tercer trabajo Terashima et al. propone el uso de algoritmos genéticos para el ordenamiento dinámico de variable en CSPs [18], el presente proyecto está estrechamente ligado con este proyecto anterior. En este proyecto se propone el uso de *Redes Neuronales Artificiales* (NNA) para la creación de la hiperheurística y más en específico la aplicación de la *Neuro-Evolución*. Burke et al. [2], utilizando heurísticas de coloreo de grafos, *Least saturation degree first*, *Largest color degree first*, *Largest degree first*, entre otras, crea una hiperheurística para la asignación de tareas, en específico en la organización de fechas de exámenes en una escuela.

La red neuronal que utilizaremos como hiperheurística será creada mediante un proceso evolutivo. Una red neuronal artificial está basada en la capacidad del cerebro de procesar información que se debe a las redes creadas por las neuronas que lo componen, algunos de los trabajos en IA estuvieron dirigidos en la creación de redes neuronales artificiales. Una red neuronal está compuesta de nodos o unidades conectadas por ligas dirigidas y cuentan con un peso y una función sigmoidal que sirve para activar una neurona [14]. Por otra parte, la Neuro-evolución (NE), la evolución artificial de redes neuronales usando algoritmos genéticos, ha mostrado ser una promesa en las tareas de aprendizaje por reforzamiento. NE sobrepasa los estándares de los métodos de aprendizaje por reforzamiento en muchas tareas [16].

1.1. Definición del Problema

Un *problema de satisfacción de restricciones* está compuesto por conjunto finito de *variables*, donde cada una cuenta con un *dominio* finito de valores posibles, adicionalmente una serie de *restricciones* entre ellas que condicionan los valores que las variables pueden tomar de manera simultánea. Solucionar un CSP consiste en encontrar una asignación consistente para todas las variables de tal manera que todas las restricciones sean satisfechas o demostrar que no es posible esta asignación, este problema es de complejidad NP completa [9]. Los métodos de solución más comunes para CSPs se basan en la propiedad que tienen de ser modelados a través de un grafo de restriccio-

nes, en el cual los nodos y arcos son las variables y restricciones respectivamente, de esta manera la solución se plantea mediante un árbol de búsqueda con un espacio de solución exponencial. Dentro de los métodos utilizados para solucionar los CSPs podemos encontrar el llamado *Backtracking* (BT), el cual es una adecuación del algoritmo primero a lo profundo. Este algoritmo consiste en asignar una variable a cada nivel del árbol de búsqueda, en caso de una asignación inconsistente, el algoritmo regresa un nivel anterior y modifica el valor de la variable y continua hasta encontrar la solución. Otro método de solución muy utilizado es el algoritmo conocido como *Foward Checking* (FC), en el cual una vez realizada una asignación, es decir, asignado un posible valor a una variable, el algoritmo modifica los dominios de las variables que están directamente ligadas por medio de una restricción a la variable eliminando los valores que faltan por instanciar que entran en conflicto con la asignación reciente. Otra propuesta de solución es el conocida como *Constraint Propagation*, este algoritmo busca propagar la información de la asignación actual a las variables sin asignar, para esto se utiliza el algoritmo *Arc Consistency* (AC). Un enfoque más inteligente del algoritmo BT es conocido como *Backjumping* (BJ), éste hace un backtracking hasta el nivel donde se produjo la asignación incorrecta [14]. Sin importar el método de búsqueda a utilizar el orden de las variables y valores juegan un papel muy importante al momento de la solución. Este orden repercute en la complejidad de la búsqueda, es por ello que encontrar métodos que ordenen de forma eficiente dichas variables es de suma importancia. Un buen orden de variables se refiere a mover asignaciones ilegales a niveles superiores del árbol de búsqueda y un buen orden de valores se refiere a mover la solución más a la izquierda del árbol para que un algoritmo basado en primero a lo profundo, como *backtracking*, pueda encontrarla más rápido.

Al hablar de orden de valores y/o variables existen dos enfoques totalmente opuestos. El primero es conocido como *ordenamiento estático*, éste consiste en establecer el orden o la secuencia de valores y/o variables al comienzo de la solución y conservarlo hasta el final. El otro enfoque es conocido como *ordenamiento dinámico* el cual consiste en elegir el valor y/o variable a asignar al momento de solucionar el CSP de acuerdo a las características actuales del mismo. Este enfoque, el dinámico, ha demostrado ser más eficiente en la solución de CSPs que el enfoque estático, es decir, que el tiempo de solución y las verificacion de restricciones se ven reducidos [5, 1]. A partir del descubrimiento de este último enfoque se ha ido creado heurísticas para este fin, entre las que destacan *Fail-First*, *Rho*, *Kappa*, entre otras; pero se ha notado que unas heurísticas son mejores en problemas con ciertas características que otras, o en diferentes estados o instancias de un problema unas demuestran mejores resultados [5, 1].

1.1.1. Hiperheurísticas

Al momento de realizar la asignación de un valor a una variable un CSP puede verse modificado, y de hecho ocurre, aunque el modelo de ordenamiento dinámico basado en heurísticas es mejor que el estático, no toma en cuenta el punto anterior, una vez determinada la heurística a aplicar, ésta se aplica a cada paso de la solución del problema. La solución es aplicar una heurística diferente a cada nivel del árbol. Se sabe que no existe una heurística universal, con la cual pueda obtenerse siempre los resultados óptimos en tiempos considerables, y se sabe también que ciertas heurísticas son preferidas bajo ciertas situaciones [4]. En esencia una hiperheurística es una búsqueda estratégica de alto nivel que selecciona de un numero de heurísticas de nivel bajo en cada punto de la búsqueda [3]. La hiperheurística administra la elección de qué heurística de bajo nivel que deberá ser aplicada en un cierto momento, dependiendo de las características de las heurísticas y la región del espacio de solución actualmente en exploración.

Una hiperheurística se define como un algoritmo de selección de una heurística sencilla para cada estado del problema, es decir, dado un conjunto H de heurísticas la hiperheurística es el algoritmo que determina de acuerdo a las características del estado actual del problema el momento de aplicar una heurística h_i particular.Es por ello que este trabajo propone el uso de Redes Neuronales como hiperheurísticas que permitan elegir de acuerdo a las características actuales de un problema la heurística que tenga un mejor desempeño al momento de elegir la variable a instanciar en comparación a la aplicación individual de heurísticas. Con estas hiperheurísticas se pretende obtener una forma eficiente de aplicar la heurística que presente un mejor rendimiento en el ordenamiento de variables dependiendo del estado del problema durante su proceso de solución.

El uso de la Neuro-Evolución en CSPs no es completamente nuevo, por ejemplo Moriarty en 1994 [10] utilizó una red neuronal evolutiva para el ordenamiento de variables en la resolución de CSPs, y Tsang en 1992 [19] utilizó una red neuronal para la solución de CSPs, pero hasta ahora nadie ha utilizado una red como hiperheurística para el ordenamiento dinámico de variables en problemas de satisfacción de restricciones. Los problemas que atacaremos son aquellos cuyo dominio es compartido entre las variables y cuentan restricciones Binarias o Unarias. Además estamos interesados en aquellos problemas que se clasifican como difíciles. Los problemas que cuenten con un nivel bajo de conectividad son problemas fácilmente solucionables y también aquellos problemas con demasiadas restricciones puesto que es fácil encontrar que no cuentan con una solución. Existe un valor crítico de restricciones donde el problema es considerado difícil [12].

1.2. Motivación

En sus orígenes la IA era considerada como ciencia de juguete [14], puesto que muchos de los algoritmos presentados eran probados en mundos creados específicamente para ellos. Un ejemplo de un CSP puede ser el acertijo de las N reinas, encontrar todas las soluciones del acertijo de las N reinas es un buen ejemplo de un problema no trivial. Por esta razón, es a menudo usado como problema ejemplo para varias técnicas de programación, incluyendo enfoques no tradicionales como CSPs. El acertijo de las N reinas consiste en poner N de éstas en un tablero de $m \times m$ de tal manera que ninguna de pueda capturar a cualquier otra usando los movimientos estándar de una reina en un juego de damas. Las reinas deben ser colocadas de tal manera que ningún par sea capaz de atacarse una a otra. De esta manera, una solución requiere que ningún par de reinas se encuentre en la misma fila, columna o diagonal. Este problema es fácilmente caracterizado como un CSP, además de que puede existir más de una manera de hacerlo, a manera de ejemplo, las reinas podrían ser consideradas como variables, las cuales tendrán como dominio las posibles posiciones en el tablero, las restricciones que ninguna represente una amenaza para alguna otra. Este acertijo en un ejemplo clásico de utilización de CSPs en investigación; pero así como éste problema podría ser considerado de juguete, los CSPs son utilizados ampliamente en la industria, problemas de asignación de recursos, diseño y manufactura de circuitos, ajustes de parámetros, etc.

Tsang como introducción a su libro *Foundations of CSP* menciona:

> Casi todos aquellos que trabajen en inteligencia artificial deberían saber sobre Problemas de Satisfacción de Restricciones, CSPs aparecen en muchas áreas[...]. Los CSPs son tan valiosos que es necesario estudiarlos por separado porque son problemas generales que cuentan con características únicas que pueden ser explotadas para llegar a soluciones [20].

Tsang en esta introducción menciona la importancia del conocimiento de CSPs, muchas áreas pueden definir sus problemas en base a las características únicas de los mismos.

La motivación de este trabajo es analizar la factibilidad de aplicar redes neuronales como hiperheurísticas y a su vez un enfoque evolutivo para determinar la mejor red neuronal, en ordenamiento dinámico de variables en CPSs, y comprobar los resultados con la aplicación de heurísticas sencillas o de bajo nivel para la misma tarea.

1.3. Objetivos

El objetivo general de este trabajo es la generación de hiperheurísticas utilizando redes neuronales para el ordenamiento dinámico de variables en la solución de CSPs considerados difíciles.

Dentro de los objetivos particulares a cumplir en este trabajo de investigación son los siguientes:

- Definir qué heurísticas existentes en la literatura utilizadas para el ordenamiento dinámico de variables serán consideradas para alimentar la red neuronal y para los resultados comparativos.
- Diseñar e implementar un modelo neuro-evolutivo para obtener una red neuronal dentro de un algoritmo determinístico de búsqueda para resolver el problema de orden dinámico de selección de variables a ser instanciadas.
- Analizar la factibilidad de resolver el problema de selección de valores a ser asignados a las variables mediante el uso de un red neuronal, pudiendo ser ésta la misma implementada para el objetivo anterior o una segunda red.
- Analizar las características de las hiperheurísticas obtenidas mediante este enfoque.
- Determinar bajo qué circunstancias el uso de las hiperheurísticas generadas mediante esta propuesta resulta en una mejora significativa en la eficiencia de la búsqueda de una solución a los CSPs.
- Analizar y comparar los resultados obtenidos por hacer uso de este enfoque con aquellos obtenidos por hacer uso de otras heurísticas propuestas en la literatura.

1.4. Suposiciones

Las instancias de CSPs estudiadas en este trabajo están restringidas a las siguientes características:

- Un conjunto de n variables.
- Un tamaño uniforme del dominio d.
- La probabilidad p_1 de que entre dos variables se presente una restricción.
- La probabilidad p_2 de que se presente un conflicto en una restricción.
- Un porcentaje elevado de las instancias serán difíciles de solucionar según la teoría de la dificultad [12].

Las suposiciones que se tomaron en cuenta para el desarrollo de este proyecto de investigación se encuentran:

- La plataforma de desarrollo es el lenguaje Java.

- El código de las subrutinas de solución de CSPs es correcto.
- Después de 80000 verificaciones de restricciones se concluye que el problema no tiene solución.

1.5. Hipótesis

Dentro de las hipótesis que se demuestran en el trabajo están las siguientes:

- El uso de evolución proporciona un enfoque viable para la creación de redes neuronales.
- El uso de un modelo basado en redes neuronales proporciona una opción viable para la creación de hiperheurísticas para el ordenamiento dinámico de variables.
- El uso de hiperheurísticas generadas por medio de un modelo basado en neuro-evolución para solucionar el problema del ordenamiento dinámico de variables presenta un mejor desempeño que la aplicación por separado de las heurísticas para ordenamiento dinámico de variables actualmente existentes en la literatura.

Las preguntas guía para la investigación, o que ésta presentación tratará de responder son:

- ¿Es posible la utilización de una red neuronal para el ordenamiento dinámico de variables dentro del contexto de un CSP?
- ¿Qué estructura o modelo de red es la que mejor ser aplica en el ordenamiento dinámico de variables?
- ¿Qué características servirán para definir la estructura de la red en la población del AG para la creación de la red neuronal en la obtención de la hiperheurística?
- ¿Existen hiperheurísticas capaces de encontrar el orden óptimo de variables en la mayoría de los problemas de satisfacción de restricciones?
- ¿Qué mejoras plantea este nuevo enfoque?
- ¿Qué mejoras plantea este modelo en comparación a las heurísticas sencillas definidas en la literatura?

1.6. Contribución

Este trabajo de investigación brinda información sobre el uso de hiperheurísticas, es decir, muestra que la aplicación de diferentes heurísticas de bajo nivel para determinar la variable a elegir al momento de solucionar un problema presenta ventajas sobre la aplicación individual de las mismas.

En función de la hipótesis esta tesis contribuye con más información sobre:

- La existencia de hiperheurísticas para el problema de ordenamiento dinámico de variables en CPSs.
- La factibilidad de emplear un enfoque neuro-evolutivo para la obtención de hiperheurísticas para el ordenamiento dinámico de variables en problemas de satisfacción de restricciones.
- La existencia de relaciones entre las diferentes características que definen el estado de un problema y una heurística de bajo nivel, así como la posibilidad de que una red neuronal las aprenda.
- La ventaja de usar redes neuronales como hiperheurísitcas, como podría ser la rapidez de obtener la heurística a aplicar una vez que la red ha sido entrenada.

1.7. Organización de la Tesis

La organización de esta tesis se presenta de la siguiente forma:

En el capítulo 2 se establece los antecedentes sobre los cuales este trabajo de investigación fue desarrollada, se habla sobre los CSPs, como su definición y métodos de solución, además sobre la neuro-evolución, así como los AGs y redes neuronales artificiales. En el capítulo 3 se presenta el modelo de solución propuesto, así como una explicación a detalle de cómo fue utilizado y fue modificado a lo largo del proyecto, para demostrar porque se considera el apropiado para el problema que se intenta resolver. Se presenta, además, lo que consideramos el estado del problema, así como las heurísticas sencillas utilizadas. En el capítulo 4 se presentan los resultados de los experimentos así como la descripción de los mismos. En el capítulo 5 se presenta, por último, las conclusiones y algunas sugerencias para trabajos futuros.

Capítulo 2

Antecedentes

En este capítulo se explicará a detalle la teoría que sustenta la propuesta y las definiciones útiles para comprender la importancia de la misma. Se hablará de lo que es un *Problema de Satisfacción de Restricciones*, así como las heurísticas existentes para la solución de los mismos y de la definición de una hiperheurística de manera general, además de lo que representa la *Neuro-Evolución*, y los dos elementos que la conforman, es decir las *Redes Neuronales* y el *Algoritmo Genético*.

2.1. Problemas de Satisfacción de Restricciones

Muchos de los problemas de las ciencias de la computación y de la inteligencia artificial (IA) pueden ser modelados como problemas de satisfacción de restricciones [8], de hecho como menciona Tsang la mayoría de los interesados en estas áreas deben saber algo de estos [20].

Como mencionamos anteriormente un CSP es un conjunto de variables $x_1, x_2, \ldots, x_n$; un dominio D de valores posibles para esas variables, y un conjunto de restricciones $c_1, c_2, \ldots, c_n$; pero de manera formal la definición de un CSP de acuerdo a Tsang es la siguiente:

Un problema de satisfacción de restricciones es una tripleta:

$$\langle X, D, C \rangle$$

donde:

- X es una serie finita de variables $x_1, x_2, \ldots, x_n$;
- D es una función que mapea cada variable en Z con una serie de objetos de tipo arbitrario:
  $D : Z \rightarrow$serie finita de objetos de cualquier tipo. Se toma D_{x_i} como el conjunto de objetos mapeados de x_i por D. Estos objetos se llamarán **valores** posibles de x_i y el conjunto D_{x_i} el dominio de x_i;

- C es el conjunto finito (posiblemente vacío) de **restricciones** en un subconjunto arbitrario de variables en X. En otras palabras, C, es un conjunto de pares de posibles asignaciones variable, valor, y posiblemente este conjunto sea vacío. De acuerdo a la literatura se usa $csp(P)$ para denotar que P es un CSP [20].

2.1.1. Métodos de Solución de un CSP

La tarea de un CSP es asignar un valor a cada variable de tal manera que se satisfagan todas las restricciones de manera simultánea, es decir, la solución a un problema CSP implica la búsqueda de las combinaciones variable/valor que satisfagan a todas las restricciones dentro de un espacio de búsqueda en el cual se encuentran las posibles soluciones, este espacio puede ser construido en forma de árbol según se seleccionen de manera secuencial las variables para ser instanciadas. En el problema del coloreo del mapa la meta es asignar colores a cada región de tal manera que ninguna región vecina tenga el mismo color. En el problema de las 8-reinas, colocar las reinas sin que éstas representen una amenaza la una a la otra.

Un CSP puede ser solucionado bajo el paradigma de *prueba y error*, en este método las asignaciones se van generando de manera aleatoria y se van evaluando en el problema para verificar la satisfacción de restricciones. La primera que satisfaga todas las restricciones es la solución [8]. Aunque este método puede parecer sencillo, el cómputo que se requeriría para realizarse es inmenso, por lo que a lo largo de la historia de los CSPs se han ido creando algoritmos con un mejor desempeño para la solución del problema. El espacio de búsqueda de un CSP es exponencial en n, el número de variables. Es por ello que buscar una forma de optimización es de suma importancia.

Backtracking

El algoritmo *backtracking* (BT) es un método más eficiente en comparación al paradigma de prueba y error, este método de solución es una adecuación del algoritmo primero a lo profundo. Los algoritmos de solución generan sucesores por considerar posibles asignaciones para una variable en cada nodo del árbol de búsqueda [14]. La operación básica en BT es elegir una variable a la vez, y considerar un valor para ella a la vez, asegurándose que la nueva asignación sea compatible con las asignaciones anteriores. Si la asignación actual viola alguna de las restricciones, entonces un valor alternativo es evaluado, siempre y cuando exista alguno disponible. Si todas las variables han sido asignadas entonces el problema ha sido solucionado. Si en algún momento de las asignaciones, ningún valor puede ser asignado sin violar alguna restricción, la asignación anterior se ve modificada, con un nuevo valor, si existiera alguno disponible. Esto se realiza hasta que todas las variables han sido asignadas o que todas las asignaciones fallen. En la figura 2.1, se encuentra un diagrama de flujo del algoritmo backtracking [20].

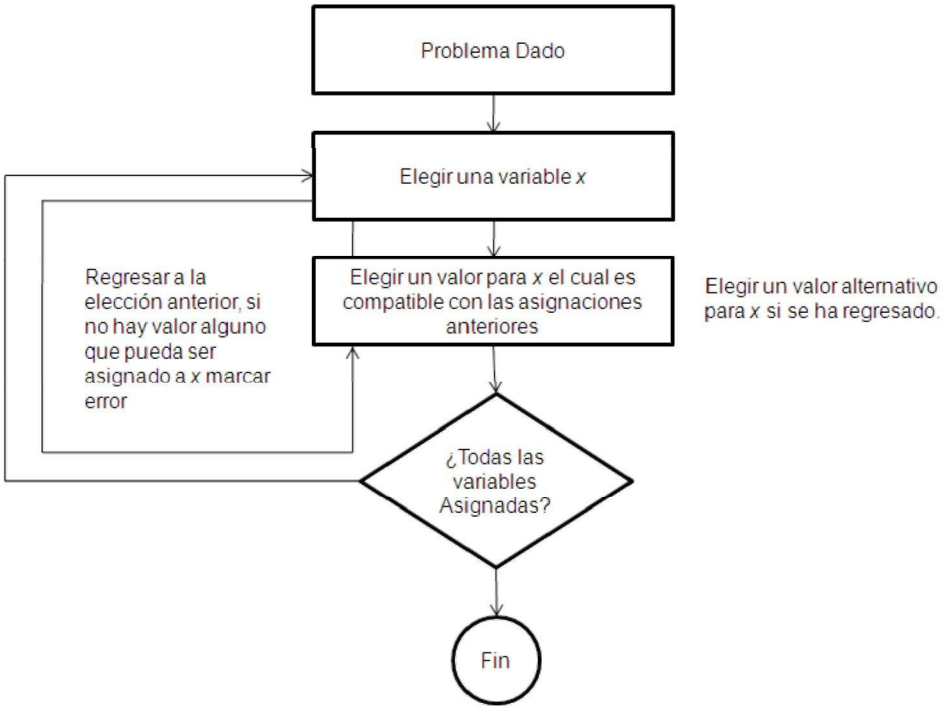

Figura 2.1: Diagrama de flujo del algoritmo backtracking cronológico

El *backtracking* es conocido también como *backtracking cronológico* puesto que el algoritmo siempre va una asignación anterior cuando no es posible continuar. Un BT inteligente, conocido como *backjumping*, mejora el desempeño del BT cronológico. El *backjumping* consiste en realizar un retroceso no a la variable anterior, sino a la variable causante del problema[20].

Aunque este algoritmo es una mejora, el tiempo de cómputo sigue siendo exponencial, este pobre rendimiento se debe, según Kumar, a que la búsqueda en diferentes partes del espacio de búsqueda falla por las mismas razones, lo que se conoce como *thrashing*. Una de las razones más comunes para este problema está relacionada con las restricciones unarias, es decir, si en el dominio de la variable x_i se encuentra un valor d_i que no satisface la restricción unaria de x_i entonces la asignación de este valor a la variable ocasionará un fallo inminente. Otro ejemplo de *thrashing* ocurre cuando dado un conjunto de variables $x_1...x_n$ asignadas en una secuencia que un valor d_i de la variable x_i, imposibilita cualquier asignación de la variable x_j, entonces cada vez que esto ocurra en *backtracking* se tendrá que realizar, esto es conocido como *consistencia de arco* [8]. El *thrashing* puede evitarse eliminando los valores de los dominios que entren en conflicto con la asignación actual, lo que nos lleva al siguiente método de solución.

Propagación de Restricciones

Forward Checking (FC) es otro método de solución el cual consiste en prever las consecuencias de una asignación. Cuando una variable X es asignada, el proceso de FC revisa las variables sin asignar Y que está conectada a X por una restricción y borra del dominio de Y los valores que entren en conflicto con la asignación de X. Aunque FC puede detectar muchas inconsistencias no puede detectar todas puesto que no busca lo suficientemente profundo [14]. *Propagación de Restricciones* es el término general para la propagación de las implicaciones de una restricción de una variable a las otras. Pensar en propagar todas las implicaciones ocuparía más tiempo de cómputo que hacer una búsqueda simple [14]. La *consistencia de arco*, (*Arc-Consistency*), es un término muy relacionado con la propagación de restricciones. La idea de *consistencia de arco* proporciona un método rápido de propagación de restricciones y es sustancialmente más fuerte que *forward checking*. Esta técnica revisa que todos los arcos definidos por las restricciones entre las variables sean consistentes; se dice que un arco (V_i,V_j) es consistente si para cada valor de x en el dominio de V_i existe un valor y en el dominio de V_j. La idea de la *consistencia del arco* es direccional, es decir, si el arco (V_i,V_j) es consistente no necesariamente el arco (V_j,V_i) lo es también. Un arco inconsistente, en algunos casos, puede hacerse consistente al borrar los valores para los cuales la definición no se aplica. Se dice que un grafo es *k-consistente* si cuenta con k arcos que satisfagan la definición de consistencia de arcos [8].

Un tercer enfoque de solución es la aplicación concurrente de BT y AC. Un nodo raíz es creado para solucionar el CSP original. En cualquier momento que un nodo es visitado, el algoritmo de propagación de las restricciones se utiliza para obtener un nivel deseado de consistencia. Si un nodo, la cardinalidad del dominio de cada variable se convierte en 1, y el CSP correspondiente es arco consistente, entonces el nodo representa una solución. Si en el proceso de propagar la restricción de un nodo, el dominio de alguna variable queda vacío, entonces el nodo es podado. De otra manera una de las variables (aquellas cuyo tamaño de dominio es 1) es seleccionada, y un nuevo CSP es creado para cada posible asignación de esta variable. Cada nuevo CSP es representado como un nodo sucesor del nodo CSP padre representado. Es importante hacer notar que el nuevo CSP es más pequeño que el CSP padre porque se necesita elegir asignaciones para una variable menos. El algoritmo de *backtracking* visita estos nodos como se acostumbra hasta que una solución sea encontrada.

2.1.2. Ordenamiento de Variables en CSPs

Los métodos anteriormente descritos son los utilizados en la solución de un CSP, determinan qué hacer cuando ocurre un fallo y algunos de ellos conocen cuales serán las implicaciones de una asignación, pero ninguno determina el orden de las variables

ni valores a evaluar. De acuerdo con los expertos, sin importar el método de búsqueda determinístico a utilizar, el orden de las variables y valores juegan un papel muy importante al momento de la solución [4, 15]. Este orden repercute en la complejidad de la búsqueda, es por ello que encontrar métodos que ordenen de forma más eficiente dichas variables es de suma importancia. Un buen orden de variables se refiere a mover asignaciones ilegales a niveles superiores del árbol de búsqueda y un buen orden de valores se refiere a mover la solución más a la izquierda del árbol para que un algoritmo basado en primero a lo profundo, como *backtracking*, pueda encontrarla más rápido. La forma de ordenar una variable se clasifican en dos categorías; el *ordenamiento estático*, (SVO por sus siglas en inglés), en éste el orden de las variables es asignado al principio de la solución y este orden se mantiene inalterable durante toda la búsqueda, la otra forma es conocida como *ordenamiento dinámico*, (DVO por sus siglas en inglés),el orden de las variables se va determinando durante el momento de ejecución. Para esta tarea se fueron creando heurísticas, las cuales son reglas que permiten orientar la ejecución de un algoritmo, en el caso especifico de ordenamiento de variables, que es el tema que nos compete, existen varias ya definidas; pero utilizaremos estas seis: *Fail-First,Fail-First Modificada* [18], *Rho*, *Kappa*, *Heurística de Saturación* y *E(n)*, las cuales se explican a continuación. Estas heurísticas están basadas en el principio de selección de la "variable más restringida"; estos algoritmos intentan fallar tan pronto como sea posible cuando están instanciando variables; lo que permite la re-instanciación de variables con otros valores, y así eliminar subregiones de búsqueda de tamaños considerables [9].

2.2. Heurísticas para el Ordenamiento Dinámico de Variables

Dentro del área de CSPs existen ciertas medidas que representan el grado de restricción de un problema, y éstas a su vez nos dan una idea de la factibilidad de solución de un CSP en específico. La medida más intuitiva es el tamaño del problema, ésta es determinada tanto por el número de variables como el tamaño de sus dominios. Este tamaño, está en base al tamaño del espacio de búsqueda de un estado del problema, el cual consiste en todas las asignaciones posibles, entonces el tamaño del espacio de búsqueda del estado se define como el producto del tamaño de los dominios de todas las variables, $\prod_{x \in Z} d_x$. Ahora se define el tamaño del problema como la suma del logaritmo base dos del tamaño del espacio de un estado.

$$N = \sum_{x \in Z} \log_2 d_x \tag{2.1}$$

Otra medida de la restricción de un problema es la *densidad de solución*. Si una restricción c tiene una fracción p_c de estar restringida, entonces existe otra fracción

$1 - p_c$ donde no lo es, es decir, existen $1 - p_c$ posibles asignaciones han ser efectuadas, entonces, la densidad de solución, ρ, es el promedio de asignaciones permitidas por todas las restricciones. p_c se obtiene al dividir el número de restricciones entre las dos variables entre el número de restricciones posibles. Un CSP con pocas restricciones tiene una densidad de solución alta, y con esto tiene mayor probabilidad de ser soluble. Si se asume independencia entre restricciones, la densidad de solución se define como:

$$\rho = \prod_{c \in C} (1 - p_c) \tag{2.2}$$

Una tercera medida de restricción se obtiene al unir el tamaño de un CSP (N), y la densidad de solución (ρ), se conoce como *número de soluciones esperadas (E(N))*,una $E(N)$ alta quiere decir que el problema está escasamente restringido, esta medida se obtiene al multiplicar las dos medidas antes mencionadas (N y ρ), quedando de la siguiente manera:

$$\begin{aligned} E(N) &= \rho 2^N \\ &= \prod_{x \in Z} d_x \times \prod_{c \in C} (1 - p_c) \end{aligned} \tag{2.3}$$

La última medida para el grado de restricción que se tiene es conocida como *Kappa*(κ), la cual surge de la combinación de la densidad de solución ($E(N)$) y el tamaño del problema (N), si κ es pequeña, el problema tiene muchas soluciones, si ésta es grande, entonces el problema tiene pocas o no tiene solución, κ es definida de la siguiente manera:

$$\begin{aligned} \kappa &= 1 - \frac{\log_2(E(N))}{N} \\ &= \frac{\log_2(\rho)}{N} \\ &= \frac{\sum_{c \in C} \log_2(1 - p_c)}{\sum_{x \in Z} \log_2(d_x)} \end{aligned} \tag{2.4}$$

2.2.1. Heurística Minimum Remaining Values o Fail-First

Para tener éxito, prueba primero donde estés más seguro de fallar, este principio conocido como el principio de *Fail-First*, aplicado al ordenamiento de variables sugiere que se intenten aquellas asignaciones que sea más probables de fallar. Esto con el fin de realizar una poda del espacio de búsqueda, es decir, que se elijan las variables más restringidas. La idea de restricción, se refiere a la variable que tenga el dominio más pequeño. Esta heurística selecciona las variables con él menor número de valores legales en su dominio. Si el dominio de la variable es grande, entonces será fácil encontrar un valor legal para asignar que en aquellas variables con dominios pequeños.

En resumen la idea consiste básicamente en tomar la variable más restringida de las que quedan sin instanciar y por lo tanto disminuir el factor de ramificación asociado a la búsqueda. Esta heurística permite encontrar de una manera rápida aquellas asignaciones que no conducen a una solución del problema, ya que se generan rápidamente ramas del árbol de búsqueda que no logran instanciar todas las variables y se produce el *backtracking*[14].

Ésta es una heurística de ordenamiento dinámico, cuando se utiliza el algoritmo forward checking (FC), ya que el orden de las variables a seleccionar se redefine cada vez que a una variable se le asigna un valor diferente, debido a que el tamaño de los dominios se ve afectado con cada asignación[9].

2.2.2. Heurística Rho

La heurística rho busca el subproblema que maximice la densidad de solución, ρ. La idea es que al seleccionar una variable se entre en el subproblema que contiene la fracción más grande de estados que son soluciones. Esto es, el subproblema con la densidad de soluciones más grande.

Esta heurística selecciona primero la variable más restringida. La idea es que al seleccionar esta variable el subproblema resultante contenga un mayor número de soluciones (densidad de solución ρ). La heurística sin embargo no toma en cuenta el dominio disponible de las variables [9].

2.2.3. Heurística E(N)

La heurística $E(N)$ busca maximizar el número de solución esperadas, esto ocurre maximizando tanto el tamaño N como la densidad de solución del subproblema a generar. Sea N, el número de soluciones de un subproblema dado, si $N = 0$, quiere decir que el subproblema no tiene solución, si $N = 1$, entonces tendrá únicamente una solución. Esta N disminuirá a medida en que se avance en el árbol de solución, pues al ir eligiendo una rama se van eliminando algunas soluciones. Es muy difícil saber el valor real de N, es por ello que se utiliza un estimado para ello, $E(N)$, entonces lo que se busca es maximizar $E(N)$ en cada estado del problema, es decir, buscamos la variable que haga que $E(N)$ se reduzca menos. Esta heurística prefiere variables muy restringidas y aquellas con restricciones severas.

2.2.4. Heurística Kappa

La heurística Kappa busca elegir la variable que minimice el factor κ del subproblema resultante. El factor κ captura la noción de que tan restringido está un problema. Los Problemas con $\kappa \ll 1$ son escasamente restringidos lo que indica que son

altamente solubles, y aquellos con $\kappa \gg 1$ son altamente restringidos lo que podría ser un indicio de que no tengan solución. De manera similar a la heurística rho (2.2.2), esta heurística intenta seleccionar una variable que deje un problema con una probabilidad alta de ser solucionable [9].

2.2.5. Heurística Grado de Saturación (Bz)

Esta heurística surgió para la solución del coloreo grafos. El *Grado de Saturación* se refiere al número de arcos o restricciones que tiene un nodo o variable. De acuerdo con la literatura la aplicación de esta heurística con el algoritmo de solución *forward checking* (FC) resulta en un algoritmo bastante competitivo en la solución de CSPs binarios [9].

2.3. Hiperheurísticas

Para muchos problemas de la vida real, una búsqueda exhaustiva para la solución es impráctico. El espacio de búsqueda puede ser inmenso, o no existe una forma de numerarlo de manera completa. Es común entonces utilizar alguna especie de heurísticas, sacrificando la garantía de encontrar la solución óptima por la ventaja de velocidad y quizás también la garantía de obtener al menos una solución con cierta calidad. Una cantidad enorme de heurísticas han sido desarrolladas para una infinidad de problemas, cada una justificada tanto de manera experimental como por argumentos basados en la clase de problemas para la cual fue creada [2]. Aunque la aparición de las heurísticas sencillas o individuales fue un gran avance, es fácil notar que cuentan con ciertas peculiaridades y limitantes. Consideremos por ejemplo la heurística Fail-First,(2.2.1), para el problema de coloreo de grafos, no ayuda en la elección del primer nodo a colorear ya que todos cuentan con tres colores en sus dominios, de ésta manera notamos que esta heurística tiene esta debilidad [2].

Al notar que las heurísticas sencillas cuentan con fortalezas y debilidades es sensato imaginar que puedan combinarse para que las fortalezas de una soporten las debilidades de otra. Una manera simple de imaginarnos una hiperheurística es como se muestra en el algoritmo 1 [2].

La idea principal en hiperheurísticas es usar los elementos de un conjunto de heurísticas conocidas para modificar el estado de un problema. La fuerza de una heurística a menudo recae en la habilidad de la misma en tomar buenas decisiones que aterricen en una excelente solución. El desventaja de este enfoque radica en que reconocer la heurística que mejor se aplique a un problema o estado no es una tarea sencilla [2]. El objetivo principal de las hiperheurísticas es incrementar el nivel de generalidad en el que los sistemas de optimización pueden operar y de esta forma dar origen a sistemas generales que puedan manipular de forma satisfactoria una amplia

Algoritmo 1 Hiperheurística sencilla

```
if Tipo de Problema (P) == p1 then
    Aplicar(heuristica1, P)
else if Tipo de Problema(P) == p2 then
    Aplicar(heuristic2, P)
else
⋮
```

variedad de problemas [21]. La forma de lograr dicho objetivo es utilizando un conjunto de heurísticas razonablemente buenas para transformar el estado del problema.

El ordenamiento dinámico de variable cuenta con algunas heurísticas aplicables [4], donde destacan las antes mencionadas. La hiperheurística tiene como objetivo determinar en que caso conviene aplicar una u otra heurística. No existe una heurística que sea la mejor y pueda ser aplicada en cada caso, es por ello que la idea de la hiperheurística suena muy tentadora dentro de los problemas de optimización.

2.4. Neuro-Evolución

La evolución artificial es un método efectivo para crear redes neuronales en tareas donde el aprendizaje por refuerzo evita los métodos supervisados como el *backpropagation*. El marco de trabajo de la evolución libra al desarrollador de generar ejemplos de entrenamiento y provee un mecanismo altamente adaptable para ambientes dinámicos. La desventaja de los métodos evolutivos, sin embargo, ha sido el gran número de evaluaciones que debe ser computada para encontrar un nivel alto de rendimiento [11, 10].

Los algoritmos genéticos han demostrado ser una herramienta muy útil en problemas de optimización, si se observa la red como una función cuyos parámetros son sus pesos, y se tiene que obtener los mejores valores para estos parámetros, es decir, los pesos, para conseguir encontrar el óptimo desempeño de la red, vemos que la *Neuro-Evolución* es factible.

Los algoritmos genéticos difieren en muchos aspectos a los métodos de optimización tradicionales, destacan cuatro diferencias [6]:

1. AGs trabajan con una codificación de los conjunto de parámetros, no con los parámetros mismos.

2. Los AGs realizan búsquedas en una población de puntos, no en un solo punto.

3. Los AGs utilizan la función objetivo, no derivadas de ellas u otro conocimiento auxiliar.

2.4.1. Algoritmos Genéticos

Los algoritmos genéticos son un modelo computacional basado en la idea de la supervivencia del individuo más apto, que es como sucede en la naturaleza. Los AGs son frecuentemente utilizados como métodos de optimización y como algoritmos de búsqueda para resolver problemas. En el AG simple, el más sencillo de los AG, se trabaja con cadenas binarias, las cuales se modifican a través del tiempo, imitando el comportamiento de los seres vivos en el proceso evolutivo. La idea es que con cada generación que transcurra los individuos sean cada vez mejores. Cada cadena binaria representa a un individuo y el conjunto de individuos recibe el nombre de población. Para la ejecución del AG primero se crea de forma aleatoria una población de individuos. Cada individuo es una cadena de bits y representa una posible solución al problema y las diferencias entre cada individuo dan lugar a que algunos sean mejores que otros. Determinar lo valioso de un individuo depende de una función de evaluación, la cual evalúa la aptitud de cada individuo y le asigna un valor numérico acorde a dicha aptitud. Posteriormente, en una fase de selección, los individuos más aptos tienen mayor probabilidad de reproducirse y por lo tanto se copian a la generación siguiente de la población. Utilizando operadores genéticos de cruce y mutación se modifican los individuos copiados a la nueva población, generando nuevos individuos que contienen la combinación de sus antecesores. El operador de cruce toma dos padres y los combina generando dos nuevos individuos, esto se realiza de forma análoga a la reproducción sexual en el proceso natural [6]. Por otra parte, el operador de mutación realiza una transformación aleatoria en alguna parte del individuo. Debido a que en la naturaleza la mutación es generalmente un proceso destructivo la probabilidad de mutación en un AG es muy baja y se utiliza como una forma de proporcionar diversidad a las soluciones [6].

Por otra parte un AG de estado estable se sustituye únicamente a un individuo en la población. No existe el concepto de generación, pues hay individuos que permanecen durante mucho tiempo; en su lugar se utiliza el término de ciclos. Este tipo de selección crea una mayor presión selectiva que la selección generacional. La idea detrás de este algoritmo se muestra a continuación: El operador de selección se mantiene de la misma manera, es decir, se realiza el proceso de selección de acuerdo a una evaluación del individuo. El proceso de mutación toma el individuo y modifica la cadena genética. En caso del operador de cruce, dos individuos son elegidos de acuerdo al operador de selección, un solo hijo es creado el cual reemplaza a un individuo de la población. Cualquier método de selección puede ser ejecutado para determinar qué individuo mutar. Existen varias estrategias de reemplazo:

- Reemplazar el peor (da una convergencia muy rápida)
- Reemplazo aleatorio

- Seleccionar el reemplazo utilizando la evaluación negativa.

En el AG de estado estable, los mejores individuos no dependen de la probabilidad para determinar si seguirán apareciendo en la siguiente población, ya que el elemento a reemplazar se selecciona de forma determinística. El AG de estado estable parece ser más rápido sin embargo puede perderse puesto que no explora el espacio de búsqueda tan bien como lo hace un AG generacional.

Se genera una población inicial de redes neuronales de manera aleatoria, se le calcula una aptitud inicial el cual es un promedio ponderado. Se eligen dos individuos de la población de manera aleatoria, de estos dos hijos se generan otros dos, los cuales vienen a sustituir a los peores individuos de la población, si es que cuentan con una mejor aptitud. Y se repite el proceso de evaluación por un número determinado de ciclos.

Explicando el proceso de un AG simple es:

1. Determinar la forma para codificar el dominio del problema o espacio de búsqueda en forma de un cromosoma.

2. Se Genera una población inicial de N individuos, la cual son posibles soluciones al problema.

3. Evaluar los individuos de la población inicial, puesto que son posibles respuestas, en la función de evaluación.

4. Seleccionar por algún método, los individuos que se convertirán en padres de la siguiente generación.

5. Seleccionar a dos individuos de manera aleatoria, estos se cruzarán con una probabilidad Pc.

6. A los nuevos individuos, se cambiará un bit de su código genético, con una probabilidad Pm.

7. Se tiene una nueva población de individuos.

8. Repetir esto hasta llegar a una condición de paro.

2.4.2. Redes Neuronales

El interés en Redes Neuronales Artificiales (RNA) surgió después de que Frank Rosenblatt en 1943 introdujeran el modelo del perceptrón, el cual era un modelo de una neurona biológica, en la figura 2.2 vemos las características de una neurona biológica y de un perceptrón, varias décadas despues Minsky y Papert publicaron su libro titulado *Perceptrons* en el cual mencionaban las deficiencias que este modelo tenía, después de

esto muchos investigadores dejaron a un lado las redes neuronales, solo unos cuantos permanecieron fieles a esta área. A principios de los ochentas el interés resurgió esto en gran medida a la nueva capacidad de procesamiento del hardware y la adecuación del algoritmo *backpropagation* [13].

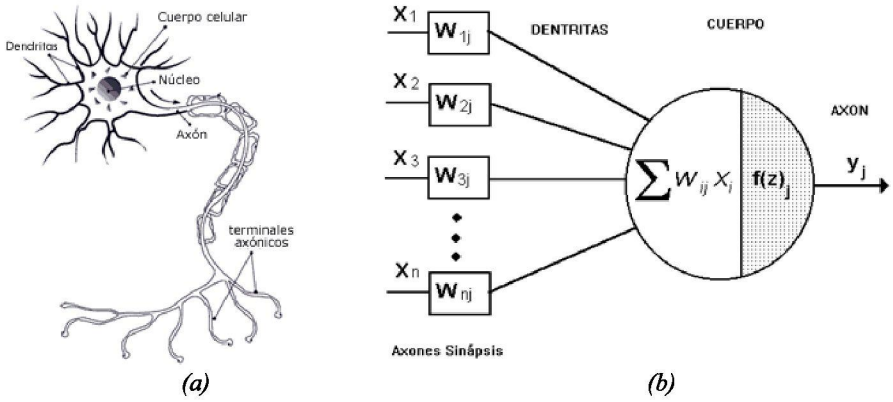

Figura 2.2: Comparación entre una neurona biológica y un perceptrón.(a) Neurona biológica, (b) Perceptrón de Rosenblatt

Las RNAs pueden ser adecuadamente caracterizadas como modelos computacionales con propiedades particulares tales como la habilidad de adaptarse, aprender, generalizar, organizar datos, y su procesamiento está basado en el procesamiento paralelo [13].

De manera un poco formal, una RNA es un sistema de procesamiento de información que tiene ciertas características en común con las redes neuronales biológicas. Las RNAs han sido desarrolladas como generalización de modelos matemáticos de cognición humana o biología neurológica, basado en las siguientes asunciones:

- El procesamiento de la información ocurre en muchos elementos simples llamados neuronas.
- Las señales se pasan entre neuronas por medio de conexiones entre neuronas llamadas sinapsis.
- Cada sinapsis tiene un peso asociado, el cual, en una típica red neuronal, multiplica la señal transmitida.
- Cada neurona aplica una función de activación (generalmente no lineal) a la entrada de la red para determinar la señal de salida.

Una red neuronal está compuesta de nodos ,o unidades, conectadas por ligas dirigidas. Cada unidad realiza una tarea simple, recibe una entrada de sus vecinos o fuentes

externas y usa esto para calcular una señal de salida, la cual es propagada a otras unidades [13]. Una liga de la unidad j a la unidad i sirve para propagar la activación a_j de j a i. Cada liga también tiene un peso $W_{i,j}$ asociado con él, el cual determina la fuerza y señal de la conexión. Cada unidad i primeramente calcula la suma de los pesos de sus entradas, en la mayoría de los casos se asume que cada unidad provee de una contribución a la entrada de otra unidad a la cual está conectada [13, 14]:

$$in_i = \sum_{j=0} W_{j,i} a_j \tag{2.5}$$

Entonces aplica una *función de activación* g a esta suma para derivar la salida:

$$a_i = g(in_i) = g\left(\sum_{j=0} W_{j,i} a_j\right). \tag{2.6}$$

La función de activación g tiene dos características, primero busca activar la unidad si el resultado de la función esta cercano a uno, y desactiva la unidad si el resultado es cercano a cero, lo segundo, la activación necesita ser *no lineal*, de otra manera la red entera se colapsa a una función lineal. La función g puede ser de varias maneras,dos ejempls de ellas pueden ser una función *step* o una función *sigmoide*, esta última tiene la ventaja de ser una función derivable.

Dentro de las RNAs es útil identificar tres tipos diferentes de unidades: unidades de entrada, las cuales reciben los datos de fuera de la neurona; unidades de salida, éstas envían los datos fuera de la red neuronal; y las unidades ocultas, aquellas donde los datos de entrada y salida se mantienen dentro de la red.

Existen dos categorías principales de las estructuras de una red neuronal: *redes feed-forward*, representa una función de sus entradas actuales, esto es, no tiene estado internos más que los pesos mismos; o *redes recurrentes*, alimenta sus entradas con sus mismas salidas. Se enfocará esta investigación a las redes neuronales feed-forwards, estas redes están organizadas en *capas*, donde las unidades solo reciben entradas de las unidades de la capa previa [14].

Trabajaremos con Redes multicapa, la ventaja de éstas es que aumenta el espacio de hipótesis que una red puede representar, no existe un método especifico para determinar el número de capas ocultas que una red debe tener.

Entrenamiento de la Red Neuronal

Un RNA tiene que ser configurada como la aplicación que para un conjunto de entrada produce un conjunto deseado de salidas. Existen varios métodos para establecer

los pesos de las conexiones. Una forma es establecer los pesos explícitamente, usando conocimiento a priori. Otra forma es entrenar la RNA alimentándola con datos de entrenamiento y permitiéndole que actualice sus pesos de acuerdo a alguna regla de aprendizaje. Para el entrenamiento se utiliza el paradigma *Aprendizaje Supervisado*, en el cual la red es entrenada al proveerle datos de entrada y salida. Estos pares de entrada y salida pueden ser provistos por un experto, o por el sistema que contiene la red [13].

El algoritmo de retropropagación del error es el algoritmo utilizando para entrenar la red, es decir, para actualizar los pesos de las neuronas, primero calcula el error en la salida para cada unidad, ajusta los pesos en la capa de salida para reducir el error y propaga los errores hacia la capa de entrada ajustando los pesos de las capas ocultas y este proceso se repite hasta que el error sea mínimo.

El entrenamiento por retropropagación del error requiere una red neuronal de topología feed-forward y puesto que es un algoritmo de aprendizaje supervisado requiere, tanto de un vector de entradas como un vector de salidas. Para un vector de entrada, el vector de salida es estimado a través de su paso sobre la red. Después que se obtenga este vector, se calcula el vector del error, al tomar la diferencia entre la salida obtenida y la salida deseada. Una función de errores de las capas de salida es propagada hacia atrás a través de la red a cada capa para ajustar los pesos en ella [7].

Para el algoritmo de retropropagación del error se cuenta con un vector $h_w(x)$ que es la salida obtenida y para cada salida se tiene un vector y que es la salida deseada. La idea clara es que el vector de errores $y - h_w$ es fácil de calcular; pero no así los errores de las capas ocultas. Para ello se tiene que retropropagar el error de la capa de salida hacia las capas ocultas. En el algoritmo 2 podemos apreciar el el pseudocódigo del algoritmo de retropropagación del error.

En la capa de salida, se actualizan los pesos de acuerdo a la siguiente formulas

$$\Delta_i = Err_i x g'(in_i) \tag{2.7}$$

$$W_{j,i} \leftarrow W_{j,i} + \alpha \times a_j \times \Delta_i \tag{2.8}$$

La idea es que el nodo oculto j es "responsable" de alguna fracción del error Δ_i en cada nodo de salida a la cual está conectado. Así los valores Δ_i son divididos de acuerdo a la fuerza de las conexiones entre los nodos ocultos y el nodo de salida y son propagados hacia atrás para proveer el valor Δ_j para la capa oculta. La regla de propagación para los valores Δ es el siguiente:

$$\Delta_j = g'(in_j) \sum_i W_{j,i} \Delta_i \tag{2.9}$$

Ahora la actualización de pesos entre la capa oculta y la de entrada es casi idéntica a la actualización con la capa de salida:

$$W_{k,j} \leftarrow W_{k,j} + \alpha \times a_k \times \Delta_j \tag{2.10}$$

Algoritmo 2 Algortimo de retropropagación del error [14]

Require: *ejemplos*, *red*

{**entradas**: *ejemplos*: un conjunto de ejemplos, cada uno con un vector de entrada x y un vector de salidas y}

{*red*: una red multicapa con L capas, pesos $W_{j,i}$, y función de activación g.}

repeat

for all e **in** *ejemplos* **do**

for all nodo j en la capa de entrada **do** $a_j \leftarrow x_j[e]$ **do**

for $\ell = 2$ **to** L **do**

$in_i \leftarrow \sum_j W_{j,i} a_j$

$a_i \leftarrow g(in_i)$

for all nodo i **en** la capa de salida **do**

$\Delta_i \leftarrow g'(in_i) \times (y_i[e] - a_i)$

for $\ell = L - 1$ **to** 1 **do**

for all nodo j **en** capa ℓ **do**

$\Delta_j \leftarrow g'(in_i) \sum_i W_{j,i} \Delta_i$

for all nodo i **en** capa $\ell + 1$ **do**

$W_{j,i} \leftarrow W_{j,i} + \alpha \times a_j \times \Delta_i$

until algún criterio de paro es satisfecho

El proceso puede ser resumido de la siguiente manera:

- Procesar los valores Δ de las unidades de salida, usando el error observado.
- Empezar con la capa de salida, repetir lo siguiente para cada capa en la red, hasta que la capa menos oculta es alcanzada:
  - Retro-propagar los valores Δ a las capas previas
  - Actualizar los pesos entre las dos capas

2.5. Trabajos Relacionados

El uso de hiperheurísticas se ha ido extendiendo entre los investigadores, el presente trabajo de investigación se encuentra estrechamente vinculado con los trabajos realizados por Terashima et al. [18] el cual se presenta, al igual que otros trabajos relacionados.

2.5.1. Un enfoque evolutivo para la generación de Hiperheurísticas

La idea detrás de las hiperheurísticas es descubrir alguna combinación de heurísticas directas para solucionar un amplio rango de problemas. Para que sea valiosa, esta combinación debería mejorar el rendimiento que la aplicación de una heurística en particular. Este proyecto presenta un enfoque evolutivo para crear hiperheurísticas generales para el ordenamiento dinámico de variables en problemas de satisfacción de restricciones. El algoritmo genético utiliza una representación de longitud variable, el cual involucra combinaciones reglas condición-acción produciendo hiperheurísticas después de un proceso de entrenamiento y pruebas [18].

2.5.2. Hiperheurísticas para Empacado Unidimensional

Farías Zarate, Claudia [21] hace uso de hiperheurísticas para el corte de material en dos dimensiones. El problema de corte es ampliamente estudiado porque cuenta con muchas aplicaciones desde el corte de ropa, metal, hasta el empacado. El problema pertenece a la clase de más difícil de problemas conocidos como NP-Hard. Dado un conjunto de piezas, el problema es generar patrones de corte para hojas de materiales, u objetos, que optimicen ciertos objetivos, tales como minimizar el desperdicio, o el número de objetos utilizados. En ésta investigación en particular se trató el corte de piezas rectangulares en dos dimensiones [21]. Varias heurísticas y enfoques de aproximación han sido propuestos que garantizan encontrar la solución óptima. Sin embargo, no ha sido posible encontrar un método confiable para solucionar todas las instancias de un problema [21]. En general algunos métodos trabajan bien para instancias particulares, pero no para todas ellas. El propósito de la investigación es explorar una alternativa nueva en el uso de enfoques evolutivos para generar hiperheurísticas para solucionar el problema de corte en dos dimensiones [21].

Una hiperheurística se usa para definir una heurística de alto nivel que controle las heurísticas de bajo nivel. La hiperheurística debe decidir cuándo y dónde aplicar cada heurística de bajo nivel, dependiendo del estado del problema dado. La elección de una heurística de bajo nivel debe depender de las características del estado del problema, tales como tiempo de CPU, número esperado de soluciones, valores de una función objetivo, etc [21]. Seleccionar una heurística particular es dinámico, y depende tanto del estado del problema como de las heurísticas previas aplicadas y del espacio de búsqueda a ser explorado en un tiempo dado [21].

2.5.3. Hiperheurísticas en Programación de Tareas

Por otro lado Edmund Burke, et al. [2], utiliza el enfoque de hiperheurísticas sobre un conjunto de heurísticas ampliamente usadas en el coloreo de grafos para la progra-

mación de tareas. Dentro del marco de las hiperheurísticas, un enfoque de búsqueda Tabú es aplicado para buscar permutaciones del grafo de heurísticas las cuales son usadas para la construcción de agendas en problemas de programación de exámenes y cursos [2].

2.6. Resumen

En este capítulo se expusieron los principales conceptos vinculados con el presente trabajo investigación. Se introduce de manera formal la definición de un CSP así como los conceptos importantes tanto para dicha definición como para los demás temas estudiados en el resto del documento. Se expuso también las heurísticas sencillas para el ordenamiento dinámico de variables y se introduce el concepto de hiperheurística. Se presenta la neuro-evolución, como un método innovador para obtener redes neuronales, así como las dos partes que la componen, por una lado se habló de los algoritmos genéticos, así como los procesos internos de los mismos, y de redes neuronales artificiales, haciendo un poco de historia de ellas, y algunos métodos de entrenamiento tales como el retropropagación del error.

Capítulo 3

Modelo de Solución

El modelo de solución presentado aquí fue resultado de una búsqueda de la generalización del trabajo presentado por Terashima et al. en [18], donde hace uso de algoritmos genéticos para crear hiperheurísticas para el mismo fin del presente. Este trabajo propone la utilización de redes neuronales como hiperheurísticas para el ordenamiento dinámico de variables en los problemas de satisfacción de restricciones. Una de las características del presente, por otra parte, es la utilización de un proceso evolutivo para obtener la mejor red neuronal de un conjunto o población de redes neuronales que persiguen el mismo propósito, más adelante se definirá qué se entiende como mejor. Como salida se obtendrá de la red neuronal la cual determinará la heurística sencilla a aplicar. La heurística determina la variable a instanciar, puesto que como se ha mencionado, el ordenamiento dinámico de variables representa una mejora en rendimiento al momento de solucionar un CSP.

3.1. Solución de un CSP

Este proceso inicia con un CSP que es representado como el *estado del CSP*, el cual se va actualizando conforme se va solucionando. El siguiente paso del proceso es elegir la variable a instanciar, es aquí donde se hace uso de la hiperheurística. La hiperheurística determinará de acuerdo al estado del CSP qué heurística debe ser aplicada para elegir la variable. Una vez determinada la variable ésta pasa al *solucionador de CSPs* el cual realiza las verificaciones de restricciones, aplicando los algoritmos de *backtracking* y *foward checking*. Este ciclo se repite hasta que el CSP haya sido solucionado o se haya determinado que no existe solución. En la figura 3.1 se pueden apreciar los pasos para solucionar un CSP. La idea principal de este trabajo es reducir el número de retrocesos, lo cual se ve reflejado en una reducción del tiempo de solución así como el cómputo requerido.

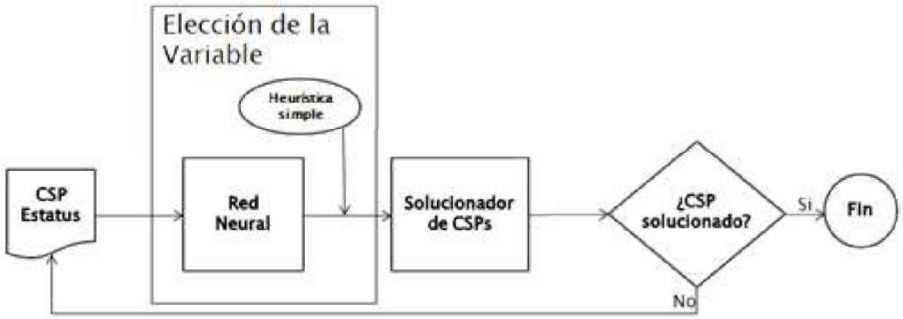

Figura 3.1: Solución de un CSP con un modulo de elección de variable

3.2. Modelo General

Este proyecto inicia con una tabla de entrenamiento que fue obtenida del proyecto de Terashima et al. [18]. El proyecto de investigación anterior genera por medio de un proceso evolutivo hiperheurísticas generales las cuales están representadas por conjunto de bloques, cada hiperheurística cuenta en promedio con 5 bloques. En la figura 3.2 podemos ver un ejemplo de la representación utilizada en el proyecto. Lo que se hace en el proyecto presente es realizar varias corridas y tomar la última población de estas para obtener el conjunto de entrenamiento que se utiliza para entrenar la red neuronal. El proceso neuro-evolutivo toma el conjunto de entrenamiento anterior y lo utiliza para entrenar la población de redes neuronales, las cuales se convertiran en las hiperheurísticas generales que se están buscando. Esto se puede apreciar de manera gráfica en la figura 3.3. El resultado de este modelo es una red neuronal entrenada la cual servirá para determinar la heurística a aplicar.

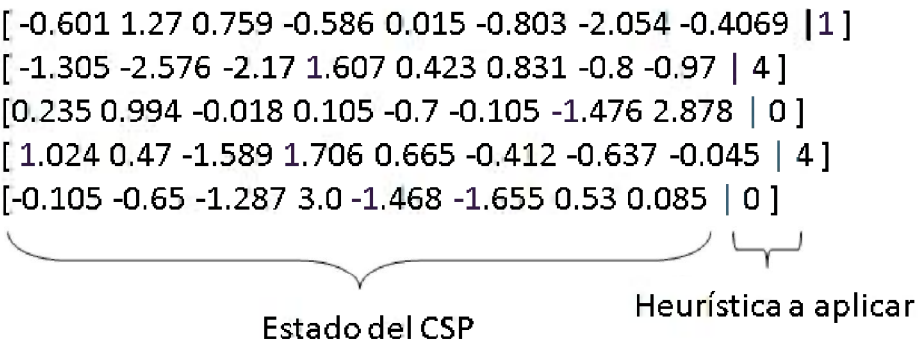

Figura 3.2: Representación de la hiperheúrística del proyecto de Terashima et al [18]

En una forma general, se presenta los procesos que están involucrados en el desarrollo del proyecto:

1. *Resolución de problemas con heurísticas simples*: Parte importante del proyecto

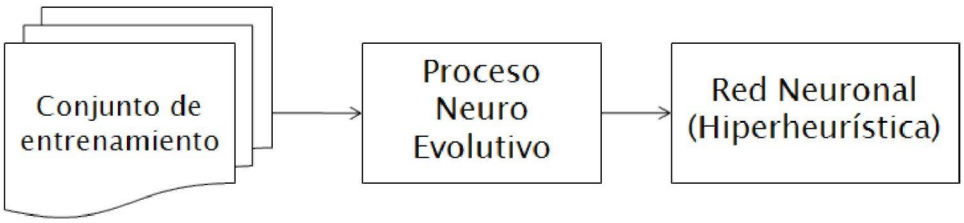

Figura 3.3: Modelo de Solución General

es encontrar una forma de comparar los resultados obtenidos con algunos resultados ya establecidos. Inicialmente se resuelven los problemas de entrenamiento y prueba con las heurísticas simples y se guardan los mejores resultados obtenidos para generar una base de información con la cual poder realizar las comparaciones y análisis pertinentes. Este estado de la solución fue obtenido del proyecto de Terashima et al. [18] por lo que no fue necesario realizarlo.

2. *Creación de la Tabla de Entrenamiento*: En este proyecto se utiliza una tabla de entrenamiento la cual contiene las entradas de la red neuronal, que son 8 factores que representan el estado de un problema. La tabla creada contiene 687 registros, cada uno con el estado de problema y la heurística simple a aplicar. Como se mencionó esta tabla fue genera después de ejecutar varias veces el sistema utilizado por Terashima et al. [18].

3. *Proceso Evolutivo para la Red Neuronal*: El algoritmo genético es el responsable de evolucionar los individuos que representan redes neuronales, que como ya se mencionó son hiperheurísticas. Estas redes neuronales están definidas mediante una serie de ocho parámetros los cuales determinan las características de la estructura de la mismas así como algunos parámetros del algoritmo de retropropagación del error utilizado en la fase de entrenamiento de las redes. Inicialmente se crea una población aleatoria de individuos, cada uno se evalúa y selecciona para dar origen a nuevos dentro de la población. Una vez creada la población ésta entra en el proceso evolutivo, el AG es ejecutado por un número de ciclos o generaciones, en cada generación la población de redes neuronales ya son hiperheurísticas con cierta aptitud. El proceso de evolución del AG se explica a detalle en las secciones subsecuentes. Después de ejecutar el proceso evolutivo se obtiene una población de hiperheurísticas, se toma la mejor la cual se utiliza para resolver las instancias de los problemas de entrenamiento y prueba.

4. *Comparación con Resultados Obtenidos*: La hiperheurística general se utiliza para resolver todas las instancias pertenecientes al conjunto de problemas de entrenamiento y prueba, los resultados obtenidos son comparados con los resultados producidos por las heurísticas simples.

3.3. Proceso Neuro-Evolutivo

En la figura 3.4 se puede observar el esquema general del proceso neuro-evolutivo. Se puede apreciar una población inicial, en este caso de redes neuronales, las cuales son entrenadas con el algoritmo *backpropagation*, después pasar por un proceso de evaluación, y por último por el proceso de cruce y mutación, repitiendo el ciclo.

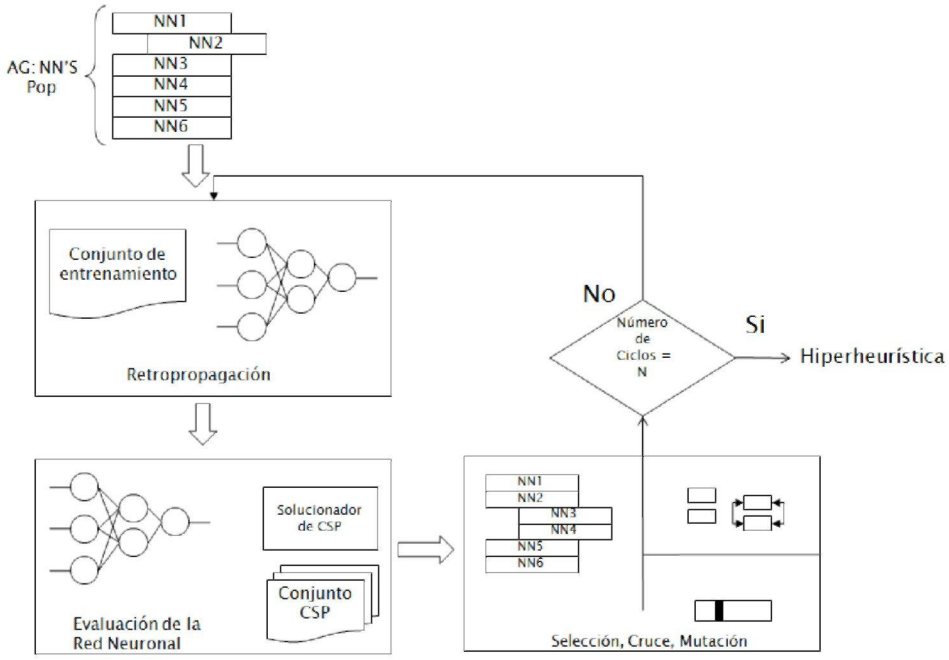

Figura 3.4: Proceso Neuro-Evolutivo

Más a detalle se tiene

1. *Generación de la Población Inicial*: Los AG están basados en la idea Darwiniana de la evolución de las especies, debido a esto parte importante del AG es su población de individuos y su representación. Se está utilizando la evolución para encontrar arquitecturas de redes neuronales, por lo que los individuos tienen codificado la estructura de la red. Para este proyecto se llegó a la conclusión, mediante experimentación, de que 30 individuos eran suficientes para obtener resultados favorables.

2. *Algoritmo de Backpropagation*: Una red antes de ser utilizada tiene que ser entrenada, existen varios métodos para realizar esta tarea; pero uno de lo más conocidos y utilizados es el conocido backpropagation, que a grandes rasgos consiste en retropropagar el error de las neuronas a las capas previas.

3. *Evaluación*: Cada individuo de la población tiene que contar con una aptitud, la cual determina qué tan valioso es el individuo en comparación con otros, una

aptitud cercana a uno en el caso de este proyecto, representa que la red neuronal aprendió como clasificar los estados que definen un problema de satisfacción de restricciones. La función de evaluación será detallada en una de las siguientes secciones ya que el éxito de un AG se encuentra en una correcta elección de la función evaluación.

4. *Selección*: Se está utilizando un AG de estado estable, entonces sólo es necesario seleccionar dos individuos para realizar el proceso de cruce y mutación.

5. *Repetir el ciclo*: Los nuevos individuos son entrenados con backpropagation y evaluados como los demás individuos de la población, y una vez entrenados y evaluados, es decir, que estos dos individuos ya posean una aptitud, estos son comparados con los existentes en la población, y pasan a sustituir a los dos peores individuos, siempre y cuando, estos, los nuevos individuos, sean mejores que los peores de la población.

3.3.1. Población Inicial: Redes Neuronales

Para encontrar la arquitectura de la red neuronal se utiliza el AG, por lo que es necesario determinar los parámetros la definen, además de algunos parámetros del algoritmo de entrenamiento.

En la figura 3.5 se observa el cromosoma general utilizado. El cromosoma consta de dos partes principales; en la primer sección se codifica el número de capas ocultas y el número de neuronas que tendrá cada una de éstas; en la segunda sección del cromosoma se encuentra algunos parámetros propios del algoritmo de backpropagation, como el número de épocas, factor de aprendizaje, factor momento entre otros.

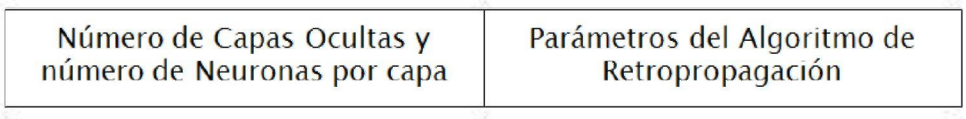

Figura 3.5: Representación del Cromosoma General

Para la representación de los parámetros se utiliza la codificación binaria. La cual es una forma de asegurar que se procesa el número máximo de esquemas. Goldberg en [6], propuso la siguiente manera para representar un número real en codificación binaria. Se escoge un número de l_i bits para representar la componente i del vector. Estos l_i bits con los valores $b_l, b_{l-1}, ..., b_2, b_1$, pueden ser interpretados como un número entero positivo.

$$x_{entero} = \sum_{j=1}^{l} b_j 2^{j-1} \tag{3.1}$$

A partir de este número entero se obtiene el valor flotante representado de la siguiente manera:

$$x_i = x_{i,entero}\left(\frac{x_{max} - x_{min}}{2_i^l - 1}\right) + x_{min} \tag{3.2}$$

Donde x_i es el valor flotante real, x_{max} el valor flotante máximo, x_{min} el valor flotante mínimo, l el número de bits utilizado para codificar el valor y $x_{i,entero}$ es la representación entera del flotante.

La fórmula para la codificación puede obtenerse despejando $x_{i,entero}$,

$$x_{i,entero} = (2_i^l - 1)\frac{x_{max} - x_{min}}{2_i^l - 1} \tag{3.3}$$

y representado este valor en binario.

Backpropagation: Tabla de Entrenamiento

Se ha estado hablando del estado del CSP, pero hasta ahora no se ha definido lo que se considera como tal. Para el desarrollo de este proyecto se consideró que el estado del problema consta de ocho factores reales, los cuales, se cree definen el momento actual del problema a ser resuelto. Estos parámetros se tomaron de Terashima et al. [18], los cuales fueron obtenidos mediante experimentación. El primer factor determina el porcentaje de variables que restan por instanciar. El segundo número es el porcentaje de los dominios que restan por instanciar. El tercero el porcentaje de restricciones que quedan en el problema. Los siguientes tres valores están relacionados con los conflictos de las restricciones entre variables que aun no han sido instanciadas. Para poder categorizar estos valores fue necesario dividir el intervalo en tres categorías diferentes, alta, mediana y poco conflictivas, para aquellos conflictos que restringen más de un 55 %, entre el 45 % y 55 % inclusive o menos del 45 % inclusive, respectivamente de asignaciones posibles. Es así como el cuarto, quinto y sexto valor son el número de restricciones alta, mediana y pocamente conflictivas respectivamente. Los últimos dos valores se refieren en especifico a los valores rho y kappa, estos valores se encuentran normalizados entre $[-3, 3]$.

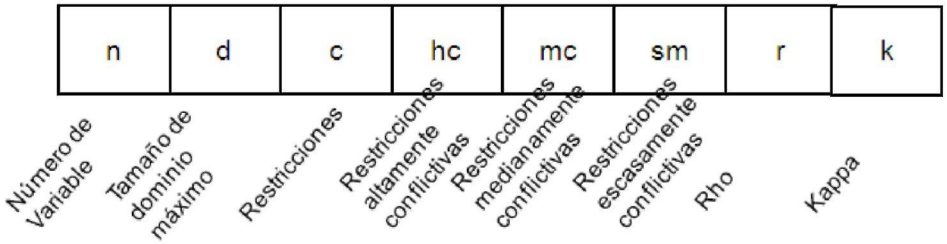

Figura 3.6: Representación del Estado de un Problema

A manera de resumen se detalla los valores antes mencionados.

- **Porcentaje de variables que quedan por instanciar (n)**:El porcentaje de variables aún por instanciar da una idea de que tan avanzado puede estar la solución. Los problemas en esta investigación fueron generados mediante un generador genérico de CSP, cuyo número máximo de variables en 25.

- **Porcentaje restante de valores en el dominio variables que quedan por instanciar (d)**: Como se ha mencionado, los problemas utilizados fueron generados utilizando un generador genérico de CSPs, el valor inicial de esta variable es 20, al momento de ir verificando conflictos y asignando valores los dominios se irán actualizando.

- **Porcentaje de restricciones en el problema (c)**: Al momento de ir asignando valores a las variables algunas restricciones van desapareciendo. Este valor da la idea de que tan restringido se encuentra aun el problema en un momento dado de la solución.

- **Porcentaje de restricciones altamente conflictivas (h_c)**: El porcentaje de restricciones en el subproblema cuyos conflictos prohíben una fracción $p_c > 0.55$ de asignaciones posibles. Debido a que el número de conflictos cambia conforme se avanza en el proceso de solución, este valor representa una buena alternativa para medir el progreso de la búsqueda de la solución.

- **Porcentaje de restricciones medianamente conflictivas (m_c)**: El porcentaje de restricciones en el subproblema cuyos conflictos prohíben una fracción entre $p_c > 0.45$ y $p_c \leq 0.55$ de asignaciones posibles.

- **Porcentaje de restricciones escasamente conflictivas (s_c)**: El porcentaje de restricciones en el subproblema cuyos conflictos prohíben una fracción $p_c \leq 0.45$ de asignaciones posibles.

- **Rho (ρ)**: Este valor es calculado como se detalla en la sección sobre heurísticas en el marco teórico.

- **Kappa (κ)**: Este valor es calculado como se detalla en la sección sobre heurísticas en el marco teórico.

Estos valores representan el estado del problema en cualquier estado de solución, lo que a su vez, representan la entrada a la red neuronal, la cual dará como salida la heurística a aplicar. Es decir a la red neuronal entran los ocho valores antes mencionados la red neuronal la cual es feed foward, tendrá como salida la heurística que determina la variable a ser instanciada.

La Tabla de entrenamiento utilizada en backpropagation se obtiene de ejecutar cinco veces el proyecto de Terashima et al. [18], con la configuración por él propuesta,

se toma la última población de individuos, los cuales son hiperheurísticas compuestas por los ocho factores antes mencionados y la heurística que conviene aplicar en este caso, cada individuo está formado por un bloque de estados-heurísticas los cuales son considerados de forma individual, es decir, cada bloque está formado por n combinación estados-heurística, estas n combinaciones forman n registros en la tabla de entrenamiento. Con esto se obtuvo una base de conocimientos de 687 registros.

3.4. Generador de Instancias de CSPs

Una parte importante del desarrollo de los experimentos es la aplicación del modelo en instancias de problemas reales. Debido a esto fue necesaria la utilización de un generador de instancias de CSPs. En este proyecto se utilizó el desarrollado por Terashima et al. en [18], el cual fue una adaptación del algoritmo presentado por Prosser en [12]. El cual se presenta a continuación.

Algoritmo 3 procedure make-random-csp(n,m,p_1,p_2)

```
c = p1 * n * (n − 1) * 0.5
while c > 0 do
  for i = 0 to n − 1 do
    while c > 0 do
      for j = 0 to n + 1 do
        while c > 0 do
          if random(1.0) < p1 and no-edge(i, j) then
            c = c − 1
            make-edge(i, j)
            for x = 1 to m do
              for y = 1 to m do
                if random(1.0) < p2 then
                  make-conflicts(i, j, x, y)
```

El generador de instancias (binarias) está caracterizado por una 4-tupla (n, m, p_1, p_2), donde n es el número variables; m es el tamaño del dominio; p_1 es la probabilidad que exista una restricción entre dos variables; y p_2 es la probabilidad de conflicto en la restricción. Con esta definición se puede entender que p_1 es la densidad del grafo de restricción y p_2 como la densidad de conflictos entre las restricciones [12]. Al crear las instancias con este algoritmo, el CSP contendrá exactamente $\frac{1}{2}p_1n(n-1)$ restricciones simétricas y tendrá aproximadamente p_2m conflictos por restricción [12]. El algoritmo empieza calculando el número de restricciones simétricas que deben ser creadas (c). Una vez calculado inicia un ciclo para generarlas. Se inicia un proceso iterativo para

crearlos conflictos con las variables existentes, dada la probabilidad p_1 se crea la restricción entre las variables siempre y cuando no exista alguna. La función **random**(1.0) genera un valor aleatorio entre 0 y 1. Una vez que el conflicto es creado con la función **make-edge**(i, j) es momento de determinar las restricciones a las cuales estará sometida este conflicto. Se inician dos ciclos anidados con el dominio de las variables y dada la probabilidad p_2 se determina si estos valores estarán restringidos, recordemos que los conflictos son unidireccionales. Para crear la restricción se utiliza la función **make-conflicts**(i, j, x, y) .De acuerdo al procedimiento de generador de instancias el CSP ($n.m$, p_1, 1) corresponde a un problema sin solución, puesto que todas las restricciones son insatisfactibles, ($n.m$, 0, p_2) es el grafo nulo, y (n,m, p_1, 0) carece de conflictos [12].

3.4.1. Conjunto de Heurísticas Simples

La finalidad de la hiperheurística es determinar qué heurística sencilla se aplicará, la cual será quien determine qué variable se instanciará, dentro del conjunto de heurísticas que se utilizaron se encuentran: *Fail-First*, *Fail-First modificada*, *Grado de saturación*, *Rho*, *E(N)*, *Kappa* y *Min-Conflicts*. Todas estas heurísticas se explicaron en la sección de marco teórico.

El valor de las heurísticas de ordenamiento asociadas a cada estado del problema se presenta en la tabla 3.4.1

Cuadro 3.1: Heurísticas sencillas utilizadas en el proyecto.

Heurística	**Valor**
Fail First	0
Rho	1
Kappa	2
E(N)	3
Bz	4
Fail First Modificada	5
Min-Conflicts	6

3.4.2. Algoritmo Genético

La parte más importante de un algoritmo genético es la elección de la función de evaluación que se busca optimizar. En el proyecto se busca reducir el número de verificaciones de restricción o los *constraint checks*, parte del desarrollo del presente, consiste en determinar qué función se optimiza. La función de evaluación que se utilizó en la mayoría de los experimentos es el promedio de dividir el número de verificación de restricciones que obtuvo la mejor heurística simple entre el número de verificación

de restricciones que tuvo la hiperheurística en cuestion. La ecuación 3.4 representa lo explicado previamente:

$$f(x) = \frac{\sum_{i}^{5+gen} \frac{CCHs}{CCHH}}{5 + gen} \tag{3.4}$$

Donde $CCHs$ es el número de verificación de restricciones que obtuvo la mejor heurística simple. $CCHH$ el número de verificación de restricciones que obtuvo la hiperheurística. gen número de ciclo del proceso evolutivo.

Selección

El proceso de selecćon es realizado utilizando en algoritmo de torneo de tamaño 2. En la seleccion de torneo de tamaño m, se mezcla la población en cuando a la posicion de los individuos. Cada individuo participa en m torneos con sus vecinos. El ganador pasa a la siguiente generación.

Cruce

La red neuronal está caracterizada con 8 parámetros, estos son representados de manera binaria dentro del cromosoma. Se utiliza cruce un punto por parámetro, por ejemplo, el parámetro $HL1_i$ (capa oculta 1 del individuo i) se va a cruzar con $HL1_j$ (capa oculta 1 del individuo j) en un punto. Esto se repite para cada uno de los parámetros que componen el cromosoma.

3.4.3. Generación de Problemas

Los problemas se clasifican para facilitar el proceso de evaluación del modelo y verificar generalidad de la hiperheurística obtenida. Como se explica en el proceso del modelo de solución se utiliza neuro-evolución para crear la hiperheurística. Se requiere de una tabla de entrenamiento para la red neuronal y un conjunto de instancias de CSPs para evaluar los individuos de la población. Se está generando un conjunto de 300 problemas por cada categoría y estos son considerados difíciles de acuerdo a lo presentado en [12].

Descripción de los Problemas

De acuerdo con Cheeseman et al. [17] existe una fase de transición asociada con los problemas NP-Completos en la cual pasan de ser fácil a difícilmente tratables, Prosser [12], en base a esto, busca la forma de determinar esa misma fase de transición dentro de los CSPs, por ejemplo, un problema que esté escasamente restringido tienden a tener múltiples soluciones, por lo que es relativamente sencillo encontrar una de éstas. Los

problemas que son altamente restringidos generalmente no poseen solución, y de igual manera no es tan complicado percatarse de este hecho. La fase de restricción crítica es aquella donde no es sencillo determinar si el problema posee o no solución, y en caso de tenerla encontrarla [4]. Un CSP está definido por cuatro parámetros, $\langle n, m, p_1, p_2 \rangle$, donde n es el número de variables, m es el tamaño del dominio, p_1 es la probabilidad de que exista una restricción entre una par de variables, y p_2 es la probabilidad de conflicto entre las restricciones. Esto es, p_1 podría ser entendido como la densidad de restricciones y p_2 como rigidez de las mismas [12].

Definimos $E(N)$ como la densidad de solución del CSP, entonces es lógico pensar que los problemas difíciles son aquellos donde exista una única solución, es decir, donde $E(N) = 1$ y esto ocurre cuando p_2 es:

$$p_2 = 1 - m^{-2/p_1(n-1)} \tag{3.5}$$

Cuando $p_1 = 0$ (o $p_2 = 0$)no hay asignaciones inconsistentes, entonces $E(N) = m^n$, es decir, todas las combinaciones de valores y variables posibles son validas. Cuando $p_2 = 1$ y $p_1 > 0$ el CSP no tiene solución, el número de soluciones se reduce rápidamente como p_2 aumenta su valor desde cero [15]

El conjunto de problemas incluyen solo instancias difíciles y dichos problemas se clasifican en tres categorías:

Clase A Estos problemas se generan con 10 variables y tamaño máximo de dominio de 20 constante. A su vez estos se clasifican en 5 categorías donde los valores de probabilidad de conflictos y restricciones se varían al momento de generar las instancias quedando de la siguiente manera A1($p_1 = 0.2, p_2 = 0.65$), A2($p_1 = 0.4, p_2 = 0.45$), A3($p_1 = 0.6, p_2 = 0.33$), A4($p_1 = 0.8, p_2 = 0.27$), A5($p_1 = 1, p_2 = 0.22$). De acuerdo con Prosser [12] con estos valores obtenemos instancias difíciles.

Clase B Estas instancias se generaron con $n = 20$ y $m = 20$, al igual que el anterior cuenta con cinco clases diferentes variando p_1 y p_2 B1($p_1 = 0.1, p_2 = 0.9$), B2($p_1 = 0.2, p_2 = 0.75$), B3($p_1 = 0.3, p_2 = 0.63$), B4($p_1 = 0.4, p_2 = 0.55$), B5($p_1 = 0.5, p_2 = 0.48$). Al igual que el grupo anterior éstas son consideradas difíciles.

Clase C Estas instancias se generaron con $n = 30$ y $m = 10$. Los valores de p_1 y p_2 se fueron variando con lo que se obtienen las siguientes clases C1($p_1 = 0.1, p_2 = 0.75$), C2($p_1 = 0.2, p_2 = 0.53$), C3($p_1 = 0.3, p_2 = 0.4$), C4($p_1 = 0.4, p_2 = 0.31$), C5($p_1 = 0.5, p_2 = 0.27$). Como en los casos previos estas instancias son consideradas difíciles de acuerdo a Prosser [12].

Conjunto de Entrenamiento

El conjunto de entrenamiento está formado por 600 instancias difíciles. 300 de estas instancias pertenecen a la clase A y 300 a la clase B. En la tabla 3.2 se puede observar

de manera detalla las características con las cuales se crearon los 600 problemas. La clase C no fue colocada, aunque son problemas diferentes a los anteriores en tamaño y complejidad, se busca determinar si la red neuronal logra generalizar los resultados de las clases A y B sobre la clase C.

Clase	n	m	p_1	p_2	**Cantidad**
A1	20	10	0.20	0.65	60
A2	20	10	0.40	0.45	60
A3	20	10	0.60	0.33	60
A4	20	10	0.80	0.27	60
A5	20	10	1.00	0.22	60
B1	20	20	0.10	0.90	60
B2	20	20	0.20	0.75	60
B3	20	20	0.30	0.63	60
B4	20	20	0.40	0.55	60
B5	20	20	0.50	0.48	60

Cuadro 3.2: Descripción de problemas del conjunto de entrenamiento

Estos problemas son utilizados para encontrar la hiperheurística general y se utilizan también para en una etapa posterior para verificar la calidad de la misma.

Conjuntos de Prueba

Se crean dos conjuntos de prueba para la verificación de la calidad de la hiperheurística general obtenida. El primer conjunto de prueba es similar al conjunto de entrenamiento, cuenta con las mismas características que estos, pero a diferencia de los anteriores estos no fueron utilizados en el proceso de entrenamiento, es decir, son problemas totalmente nuevos para la hiperheurística. Los resultados obtenidos con este conjunto de prueba representan una medida de eficiencia ante problemas nunca antes vistos. Este conjunto cuenta con 400 problemas, 200 de la clase A y 200 de la clase B. En la tabla 3.3 se puede ver las características de los parámetros con los cuales son generadas las instancias de este conjunto.

El segundo conjunto de prueba, está compuesto por 250 problemas todos estos de la clase C, y fueron creados con las siguientes características. En la tabla 3.4 se puede ver las características de los parámetros con los cuales son generadas las instancias de este conjunto.

Estos problemas, al igual que los del conjunto anterior, son totalmente nuevos para la hiperheurísticas. Estas instancias son utilizados para probar la hiperheurística ante instancias con parámetros diferentes a los utilizados en los problemas de entrenamiento.

Clase	n	m	p_1	p_2	Cantidad
A1	20	10	0.20	0.65	40
A2	20	10	0.40	0.45	40
A3	20	10	0.60	0.33	40
A4	20	10	0.80	0.27	40
A5	20	10	1.00	0.22	40
B1	20	20	0.10	0.90	40
B2	20	20	0.20	0.75	40
B3	20	20	0.30	0.63	40
B4	20	20	0.40	0.55	40
B5	20	20	0.50	0.48	40

Cuadro 3.3: Descripción de problemas del conjunto de prueba I

Clase	n	m	p_1	p_2	Cantidad
C1	30	10	0.10	0.75	50
C2	30	10	0.20	0.53	50
C3	30	10	0.30	0.40	50
C4	30	10	0.40	0.30	50
C5	30	10	0.50	0.27	50

Cuadro 3.4: Descripción de problemas del conjunto de entrenamiento

3.5. Resumen

En este capítulo se describió de manera detallada el modelo solución que se propone. Primero se presenta el proceso de manera general para después explicar con más detalle cada etapa del proceso. Se explicó también en qué parte de la solución de un CSP se coloca la hiperheurística. Se definen los conceptos que se requieren para el mejor entendimiento del capítulo y del proyecto en sí.

Capítulo 4

Experimentos Preliminares

En este capítulo se describen los experimentos preliminares del modelo. Estos experimentos dan una idea de cuál es el comportamiento del mismo y a su vez permite encontrar los puntos que requieran ser fortalecidos. Los experimentos además de probar el modelo, servirán para ajustar la función evaluación del mismo. Los resultados obtenidos con el modelo son comparados con la aplicación individual de las heurísticas sencillas para el ordenamiento dinámico de variables en CSPs.

4.1. Parámetros del Algoritmo Genético

Los parámetros del algoritmo genético fueron establecidos de manera empírica después de una serie de experimentos y pruebas. Dentro de los factores que se tomaron en cuenta al momento de elegir estos valores fueron el tiempo de ejecución del algoritmo así como la calidad de la hiperheurística obtenida. Los parámetros se presentan a continuación: El tamaño del población fue de 20, se ejecuta por 100 ciclos con una probabilidad de cruce de 1.0, y una probabilidad de mutación de 0.01. La codificación de los genes fue paramétrica y cada elemento del gen tuvo una representación binaria.

4.1.1. Representación de la Red Neuronal

En la población de individuos se encuentran caracterizados los parámetros de la estructura de la red neuronal. Los elementos que contienen son los siguientes y pueden ser vistos en la figura 4.1.

- *Elementos de la Capa Primer Oculta*: No existe forma alguna de determinar cuántas neuronas colocar en cada capa oculta, por lo que se determinó de manera empírica que el máximo número de elementos que tendrá esta capa es de $2n - 1$ donde n es el número de entradas de la red. Por ser la primera capa, tiene como mínimo una neurona, con esto se asegura que ésta exista. Este parámetro utiliza cuatro bits para su codificación.

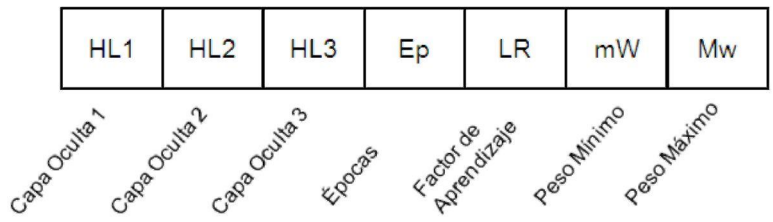

Figura 4.1: Representación del cromosoma de la red neuronal.

- *Elementos de la Segunda y Tercera Capa Oculta*: De igual manera que la primer capa tendrá un máximo de $2n - 1$ neuronas, pero a diferencia de la primera esta podrá tener cero elementos, es decir, esta capa puede ser omitida. Al igual que el parámetro anterior este parámetro utiliza cuatro bits para su codificación (cada capa).

- *Épocas*: Como se ha mencionado, se utilizará el algoritmo de backpropagation para el entrenamiento de la red, por lo que es necesario definir el número de épocas el cual será ejecutado, este parámetro está limitado por un mínimo de 687 y un máximo de 1500. No existe una forma de determinar el número de épocas por lo que el valor fue determinado sólo de manera empírica. Para representar este valor se utilizó 10 bits.

- *Límites de los Pesos*: Los parámetros que define el estado de un CSP se encuentran normalizados entre los valores -3 y 3, es por ello que el peso mínimo y máximo de las sinapsis se encuentra también entre estos valores respectivamente. Ambos parámetros fueron representados o caracterizados por 20 bits.

- *Factor de Aprendizaje*: Por último se encuentra codificado el rango de aprendizaje de la red, este valor toma valores entre 0.1 y 0.8, este valor determina que tan rápido aprende la red. Se utiliza 20 bits para su codificación.

A manera de resumen tenemos el siguiente diagrama de la representación de los individuos de la población del AG, En la figura 4.1 se aprecian los 7 parámetros que se utilizaron para describir la estructura de la población de redes, así como algunos parámetros del algoritmo de backpropagation, se debe hacer notar que este modelo de cromosoma fue utilizado en los experimentos preliminares, el cual sufrió algunas ligeras variaciones en los experimentos finales.

4.2. Experimentos y Análisis de Resultados

Una vez obtenida la hiperheurística general mediante el proceso evolutivo, su calidad fue verificada utilizándola para resolver todos los problemas del conjunto de

entrenamiento. Los resultados se presentan en términos de reducción o aumento de verificaciones de consistencia con respecto al mejor resultado de las heurísticas simples. Después de resolver los problemas del conjunto de entrenamiento utilizando la hiperheurística general.

Se ejecuta en varias ocasiones este modelo cambiando la función de evaluación para encontrar aquella con la cual se obtengan mejores resultados, de acuerdo a las *verificación de restricciones*. El objetivo de estos experimentos fue determinar la mejor función evaluación tanto para las redes neuronales así como para el algoritmo genético en sí.

4.2.1. Experimento: Creación de Una Red Neuronal de Manera Empírica

Se intenta crear una red neuronal capaz de aprender la tabla de entrenamiento, los parámetros utilizados se presentan en la tabla 4.1. Estos valores fueron los que dieron el mejor resultado después de una serie de intentos, todos estos realizados de manera empírica.

Dato	**Valor**
Capa Oculta 1	4 neuronas
Capa Oculta 2	10 neuronas
Momento	.4
Épocas	1200
Factor de Aprendizaje	0.4
Mínimo Peso	-3
Máximo Peso	3

Cuadro 4.1: Detalles de la Hiperheurística obtenida

Esta red como se ha mencionado obtuvo mejores resultado en comparación a las creadas de la misma manera, en la siguiente table 4.2 se presentan los resultados que se obtuvieron.

Lo que se puede observar en la tabla anterior fue que la red neuronal en la mayoría de los casos requiere de más del 15 % extra de verificaciones. En el conjunto de entrenamiento en el 91 % de las instancias se requiere de un extra en el número de verificaciones y apenas en un 4 % se logró comportarse como la mejor heurística. En el caso de la clase B del mismo conjunto, el comportamiento es un tanto similar, en el 85 % se requirió de más del 15 % extra en verificaciones mientras que solo en un 10.67 % se comportó como la mejor heurística sencilla. En ninguna de las clases hubo alguna mejora significativa.

		Reducción		Igual	Incremento	
Conjunto	**Clase**	$> 15\,\%$	$15\,\%$ **a** $3\,\%$	$\pm 3\,\%$	$3\,\%$ **a** $15\,\%$	$> 15\,\%$
Entrenamiento	A	0.00	0.00	4.00	5.00	91.00
Entrenamiento	B	0.00	0.00	10.67	4.33	85.00
Prueba I	A	0	0	3.5	6	90.5
Prueba I	B	6.00	0.50	8.00	5.00	80.50
Prueba II	C	0.00	0.00	12.80	3.20	84.00

Cuadro 4.2: Resultados de la aplicación de una red neuronal creada de manera empírica

En el conjunto de prueba I, los resultados se mantuvieron, en el $90.5\,\%$ de las instancias se requiere del $15\,\%$ extra de verificaciones para la clase A, y no se logró una mejora significativa en la reducción del número de verificaciones en alguna instancia. En el caso de la clase B del mismo conjunto la mejora significativa se presentó en el $6\,\%$ de las instancias, pero en el $80.5\,\%$ de ellas se requirió de más del $15\,\%$ extra de verificaciones.

Para el conjunto de prueba II, la hiperheurística se comportó como la mejor heurística sencilla en el $12.8\,\%$ de los casos, mientras que requirió de un $15\,\%$ extra en el $84\,\%$ de las instancias.

En esta investigación se presenta el número de verificaciones de restricciones expresado como un logaritmo para ser congruentes con [12]. En la figura 4.2 se observa lo que se muestra en la tabla anterior. El comportamiento de la hiperheurística no es competitivo contra la heurística simple, los resultados demuestran que la red neuronal no fue capaz de aprender el set de entranemiento y generalizarlo. Es claramente visible en la gráfica que el comportamiento de la hiperheurística queda muy por debajo del esperado.

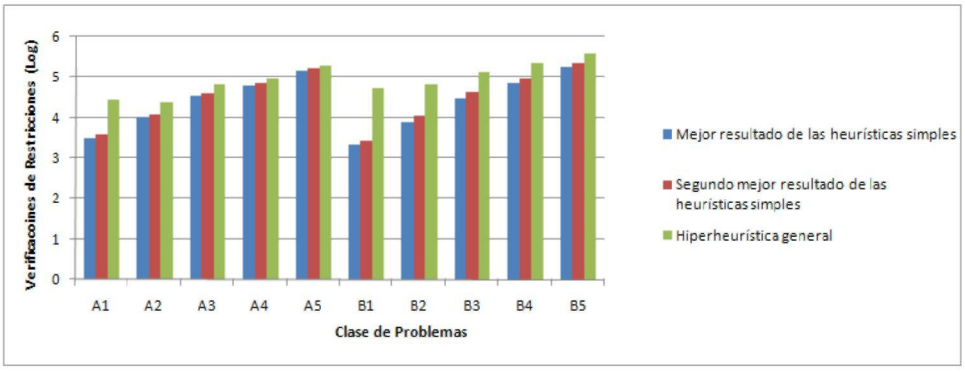

Figura 4.2: Resultados de la aplicación de la red neuronal empírica: Conjunto de Entrenamiento

En la figura 4.3 se ve la comparación entre la hiperheurística y el promedio de las

verificaciones de restricciones de las heurísticas sencillas utilizadas. Se puede observar con claridad que la hiperheurística obtenida es mejor en todos los casos que el promedio, esto se debe a que como se ha mencionado no existe una heurística que sea la mejor para todas las instancias. En algunos casos las heurísticas sencillas requieren de demasiadas verificaciones para solucionar el problema.

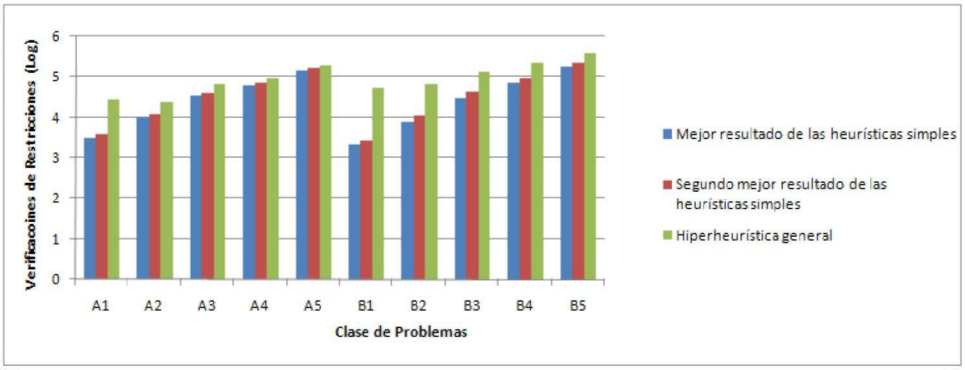

Figura 4.3: Resultados de la aplicación de la red neuronal empírica: Conjunto de Entrenamiento, Comparación con el Promedio de las Heurísticas Sencillas

Para observar el comportamiento de la red neuronal con los conjuntos de Prueba I y Prueba II, ver las gráficas el apendice B, de manera especifica las gráficas B.1, B.2, B.3 y B.4.

4.2.2. Experimento I

El conjunto de entrenamiento fue de 600 elementos de Tipo A y B, cada categoría de 60 elementos. Cada generación se genera una población mediante selección de torneo, se evalúa la hiperheurística generada con 5 problemas del conjunto de entrenamiento. En este experimento, el conjunto de prueba es el mismo que el conjunto de entrenamiento.

La forma de evaluar a los individuos que se utilizó fue la siguiente: A la población se le presentan 5 problemas al azar del conjunto de 600 problemas con que se cuenta y se obtiene el número de *constraint checks*, los cuales son comparados con la mejor heurística sencilla, y con la diferencia de los 5 problemas se obtiene un promedio, lo cual se consideró la aptitud del individuo.

Una hiperheurística puede ser clasificada de tres maneras de acuerdo a los resultados obtenidos. Puede reducir el número de constraint checks, dejarlo igual o incrementarlo. Para este proyecto, se considero que existe una mejora significativa si la reducción es superior a un 15 %. Una mejora simple si es mayor a un 3 %. Que no existe diferencia entre la hiperheurística y la aplicación de una heurística simple si la diferencia es menor a un ± 3 %. También se presentan los casos cuando la heurística simple es mejor que la hiperheurística, y de igual manera se presentan cuando la diferencia es

mayor a un 15 % y mayor a un 3 %. Esta clasificación se utiliza para ser congruentes con la investigación previa [18] y se pueda realizar las comparaciones pertinentes. Los parámetros del algoritmo genético utilizado pueden verse en la tabla 4.3

Dato	**Valor**
Población	20 individuos
Ciclos	100
Probabilidad de Cruce	1
Probabilidad de Mutación	0.01
Algoritmo Genético	Generacional

Cuadro 4.3: Detalles del algoritmo genético utilizado

La red neuronal obtenida cuenta con 3 capas ocultas con 3, 4, 3 capas ocultas respectivamente, el algoritmo de *backpropagation* tuvo 892 épocas. El factor de aprendizaje fue de 0.580 y el peso se encuentra dentro del intervalo $[-1.52, -0.3886]$. Estos datos pueden verse en la tabla 4.4.

Dato	**Valor**
Capa Oculta 1	3 neuronas
Capa Oculta 2	4 neuronas
Capa Oculta 3	3 neuronas
Épocas	892
Learning Rate	0.5807733038
Mínimo Peso	-1.528077
Máximo Peso	0.388654674
Aptitud Obtenida	0.475584873

Cuadro 4.4: Detalles de la Hiperheurística obtenida

Los resultados que se obtuvieron para este experimento se pueden apreciar en la tabla 4.5,

Se puede observar en la tabla que la hiperheurística en el conjunto de entrenamiento tanto para problemas de clase A y B la mejora no es muy notoria. Para el conjunto A sólo en un 1 % de las instancias se logra una mejora significativa, mientras que en el 74.67 % de las instancias los resultados empeoraron considerablemente. En el conjunto de entrenamiento para los problemas de la clase B tampoco hay mejora significativa, solo en un .67 % de los casos se logró la reducción del más del 15 % de las verificaciones de restricciones. Mientras que el 86 % de las instancias la hiperheurística necesita más del 15 % de verificaciones de restricciones.

Cuando se realizaron las pruebas en el conjunto de prueba I se obtuvieron datos interesantes, por ejemplo, la mejora significativa se logra en el 1 % de las instancias de la

		Reducción		Igual	Incremento	
Conjunto	**Clase**	$> 15\,\%$	$15\,\%$ **a** $3\,\%$	$\pm 3\,\%$	$3\,\%$ **a** $15\,\%$	$> 15\,\%$
Entrenamiento	A	1.00	0.33	3.33	20.67	74.67
Entrenamiento	B	0.67	1.33	4.33	7.67	86.00
Prueba I	A	1.00	1.00	5.50	20.00	72.50
Prueba I	B	7.00	2.50	7.00	9.50	74.00
Prueba II	C	1.60	1.20	12.00	5.60	79.60

Cuadro 4.5: Resultados del Experimento I

clase A, pero en un $72.5\,\%$ se requiere de $15\,\%$ extra en verificaciones de restricciones. Para la clase B del mismo conjunto se logra la mejora significativa en el $7\,\%$ de los casos; pero nuevamente se requiere de más del $15\,\%$ de verificaciones en el $74\,\%$ de las instancias. El conjunto de prueba I contiene problemas que son totalmente nuevos para la hiperheurística.

En conjunto de prueba II, que cuenta con problemas más difíciles de acuerdo a los publicado por Prosser [12], la hiperheurística obtiene los siguientes resultados: En un $1.6\,\%$ se tiene una mejora significativa reduciendo más del $15\,\%$ de las verificaciones de restricciones, en un $12\,\%$ se comporta como la mejor heurística y en un $79.60\,\%$ incrementa las verificaciones.

En la figura 4.4 se observa lo que se muestra en la tabla anterior. El comportamiento de la hiperheurística no es competitivo contra la heurística simple, los resultados demuestran que la red neuronal no fue capaz de aprender el set de entrenamiento y generalizarlo. Es claramente visible en la gráfica que el comportamiento de la hiperheurística queda muy por debajo del esperado. Sin embargo también se nota un comportamiento un poco confuso en la sección de los problemas A5, en esa sección se obtuvo que la hiperheurística se comportó en un $60\,\%$ de los casos como la mejor heurística.

Figura 4.4: Resultados del Experimento I: Conjunto de Entrenamiento

En la figura 4.5 se ve la comparación entre la hiperheurística y el promedio de las

verificaciones de restricciones de las heurísticas sencillas utilizadas. Se puede observar con claridad que la hiperheurística obtenida es mejor en todos los casos que el promedio, esto se debe a que como se ha mencionado no existe una heurística que sea la mejor para todas las instancias. En algunos casos las heurísticas sencillas requieren de demasiadas verificaciones para solucionar el problema.

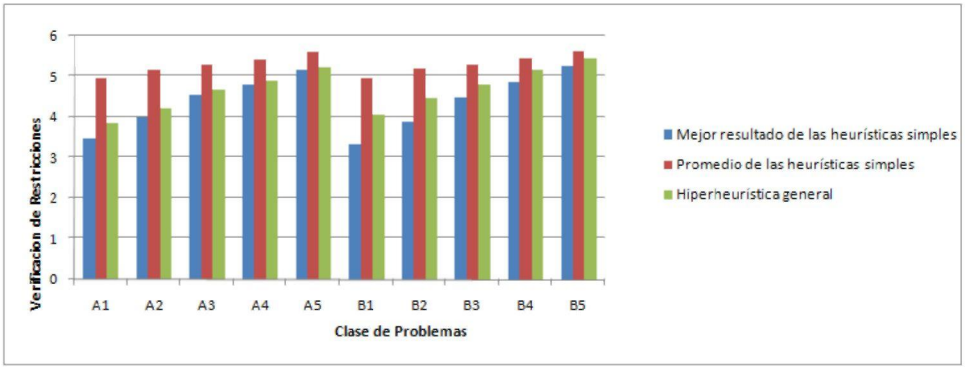

Figura 4.5: Resultados del Experimento I: Conjunto de Entrenamiento, Comparación con el Promedio de las Heurísticas Sencillas

En el apéndice B se puede observar las gráficas respectivas al conjunto de prueba I y II.

La hiperheurística fue obtenida aplicando un Algoritmo Genético Simple, el tiempo de entrenamiento fue demasiado. Al momento de realizar el cruce de los individuos, la población completa se cruzaba y mutaba, con ello, cada generación o ciclo la población es evaluada en su totalidad de la forma anteriormente explicada, es decir, no era un algoritmo genético de estado estable. El individuo evaluado fue el que tuvo la mejor aptitud en todas las generaciones, el cual fue encontrado en la primer generación con una aptitud de .475584873. Al ser encontrado en la primera generación las generaciones siguientes fueron en realidad una pérdida de esfuerzo y tiempo computacional.

4.2.3. Experimento II

En este experimento la función de evaluación es modificada. Se utiliza el porcentaje de aciertos obtenidos por la red neuronal. La Red neuronal se entrena utilizando un conjunto de entrenamiento con 687 registros, donde cada uno representaba un estado del problema, así como la heurística a aplicar. Como se ha mencionado este set de estados se obtuvo de [18]. El set de prueba, también se obtuvo de la misma manera, en este caso, es un conjunto de 45 estados, de los cuales se toman 20 de manera aleatoria y se obtiene la aptitud del cromosoma de la red neuronal. Cada generación se genera una población mediante selección de torneo. En este experimento, el conjunto de prueba es el mismo que el conjunto de entrenamiento para el algoritmo genético.

Dato	Valor
Población	20 individuos
Ciclos	100
Probabilidad de Cruce	1
Probabilidad de Mutación	0.01
Algoritmo Genético	Generacional

Cuadro 4.6: Detalles del algoritmo genético utilizado

Los parámetros del algoritmo genético utilizado pueden verse en la tabla 4.6
Los detalles de la hiperheurística obtenida pueden ser analizados en la tabla 4.7.

Dato	Valor
Capa Oculta 1	13 neuronas
Capa Oculta 2	2 neuronas
Capa Oculta 3	11 neuronas
Épocas	949
Learning Rate	0.5959497419053641
Mínimo Peso	−1.211731481197346
Máximo Peso	0.601934682789401
Aptitud Obtenida	0.4992283950617284

Cuadro 4.7: Detalles de la Hiperheurística obtenida

En el apéndice B se puede observar las gráficas respectivas al conjunto de prueba I (B.5) y II (B.7).

		Reducción		**Igual**	**Incremento**	
Conjunto	**Clase**	$> 15\,\%$	$15\,\%$ **a** $3\,\%$	$\pm 3\,\%$	$3\,\%$ **a** $15\,\%$	$> 15\,\%$
Entrenamiento	A	13.67	4.00	3.67	4.00	74.67
Entrenamiento	B	15.33	4.00	2.67	5.33	72.67
Prueba I	A	0.00	0.00	3.50	6.00	90.50
Prueba I	B	6.00	0.50	8.00	5.00	80.50
Prueba II	C	0.00	0.00	12.80	3.20	84.00

Cuadro 4.8: Resultados del Experimento II

La hiperheurística obtiene resultados interesantes, mas no buenos. Para la clase A del conjunto de entrenamiento la hiperheurística es mejor que la aplicación de la heurística simple en 13.67 % de las instancias, pero se requiere de más del 15 % de verificaciones en el 74.67 % de ellas. Para la clase B los resultados son similares en

el 15.33 % de las instancias la hiperheurística es considerablemente buena, pero en el 72.67 % no lo es.

Los resultados continúan igual en el conjunto de prueba I, en un pequeño porcentaje 3.50 % se comporta como la mejor heurística simple, mientras que en el 90.50 % no lo obtiene, esto para la clase A. Para la clase B en el 6 % de los CSPs la hiperheurística reduce es más del 15 % las verificaciones, pero en un 80.50 % se requiere de más del 15 % de las verificaciones.

En el caso del conjunto de prueba II, los resultados empeoran. En el 84 % la hiperheurística requiere de un 15 % extra en verificaciones de restricciones, y en ninguna instancia se logra una mejora significativa. Solo en un 12.8 % se comporta como la mejor.

De igual manera que el experimento anterior el Algoritmo Genético que se utiliza no es de estado estable. El mejor individuo obtenido aparece en la generación 0.

La gráfica 4.6 muestra que el comportamiento de la Hiperheurística no es competitivo en comparación de la heurística sencilla.

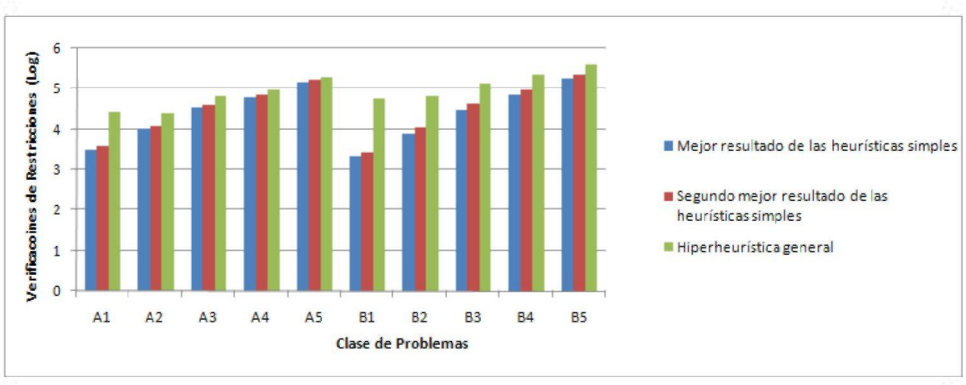

Figura 4.6: Resultados del Experimento II: Conjunto de Entrenamiento, Comparación con el Promedio de las Heurísticas Sencillas

En la gráfica 4.7 se muestra la comparación de la hiperheurística con respecto al promedio de las heurísticas sencillas. En este caso se ve que la hiperheurística si tiene un comportamiento competitivo; pero como se ha mostrado, existe una heurística simple en la mayoría de los casos (no la misma) que es mejor que ella.

En el apéndice B se puede observar las gráficas respectivas al conjunto de prueba I B.9 y II B.11.

4.2.4. Experimento III

En este experimento, se modificó la función de selección del algoritmo genético, se tomarían dos individuos al azar y se cruzarían, estos nuevos individuos ocuparían el lugar de los dos peores individuos de la población, solo en caso de ser mejores que ellos; pero si solo uno de los nuevos individuos es mejor que alguno de los individuos de

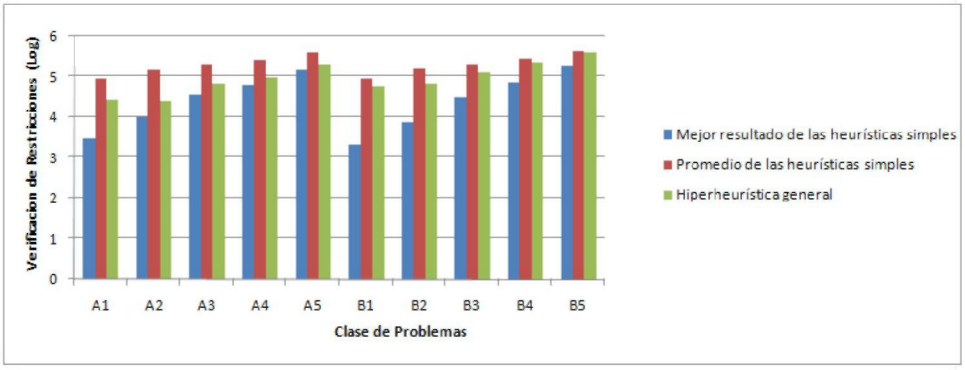

Figura 4.7: Resultados del Experimento II: Conjunto de Entrenamiento, Comparación con el Promedio de las Heurísticas Sencillas

la población el nuevo individuo ocupará el lugar del peor individuo de la población. Al momento de evaluar los individuos, estos se evaluarán el número de generaciones que llevan, es decir, los nuevos individuos tendrán el mismo número de evaluaciones que los individuos que aún permanecen en la población, pues, en cada generación la población es evaluada con un problema más. Los resultados que se obtuvieron se presentan a continuación.

Esta hiperheurística se obtiene aplicando un algoritmo genético de estado estable, con una población de 20 individuos, después de 100 ciclos. Los parámetros del algoritmo genético utilizado pueden verse en la tabla 4.9

Dato	**Valor**
Población	20 individuos
Ciclos	100
Probabilidad de Cruce	1
Probabilidad de Mutación	0.01
Algoritmo Genético	Estado Estable

Cuadro 4.9: Detalles del algoritmo genético utilizado

La red neuronal que se obtuvo cuenta con 2 capas ocultas 6 neuronas en la primer capa y 11 en la segunda, se realizaron 741 épocas en el proceso de *backpropagation*. El factor de aprendizaje fue de 0.6080 aproximadamente. El valor mínimo en el peso que pudo haber tomado es de -1.0808 aproximadamente y 0.32 como valor máximo. La aptitud que tuvo esta neurona fue de 0.561663. A manera de resumen se presentan estos resultados en la siguiente tabla 4.10.

Los resultados de la aplicación de esta hiperheurística se ven en la tabla 4.11.

Esta función de evaluación tampoco obtiene buenos resultados. En el caso del entrenamiento en las instancias de la clase A el incremento se da en el 89.33 %, mientras

Dato	Valor
Capa Oculta 1	6 neuronas
Capa Oculta 2	11 neuronas
Capa Oculta 3	3 neuronas
Épocas	741
Learning Rate	0.6080
Mínimo Peso	−1.0808
Máximo Peso	0.32
Aptitud Obtenida	0.561663

Cuadro 4.10: Detalles de la Hiperheurística obtenida

		Reducción		**Igual**	**Incremento**	
Conjunto	**Clase**	$> 15\,\%$	$15\,\%$ **a** $3\,\%$	$\pm 3\,\%$	$3\,\%$ **a** $15\,\%$	$> 15\,\%$
Entrenamiento	A	0.00	0.00	5.67	5.00	89.33
Entrenamiento	B	0.00	0.00	5.67	6.67	87.67
Prueba I	A	0.00	0.00	4.50	2.50	93.00
Prueba I	B	2.00	1.50	6.50	7.00	83.00
Prueba II	C	1.60	1.20	12.00	5.60	79.60

Cuadro 4.11: Resultados del Experimento III

que en B en el 87.67. En ninguna instancia del conjunto de entrenamiento se logra mejora alguna.

La hiperheurística en el conjunto de Prueba I, se comporta también de una forma poco competitiva. En la clase de CSP de tipo A en el 93 % se requiere de más del 15 % extra de verificaciones mientras que en la clase B esto se da en el 83 %. Pero hay una mejor significativa en el 2 % de los problemas.

En el conjunto de prueba II, se logra una mejora significativa en el 1.6 % de los problemas, mientras que se requiere de más de un 15 % extra de verificaciones en el 79.6 % de las instancias.

En la figura 4.8 se observa lo que se presenta en la tabla 4.11. Existe una heurística simple que ofrece un mejor rendimiento que la hiperheurística.

Cuando comparamos con el promedio, vemos que la hiperheurística si compite contra el promedio, esto puede ser analizado en la figura 4.9. Se puede observar con claridad que la hiperheurística obtenida es mejor en todos los casos que el promedio, esto se debe a que como se ha mencionado no existe una heurística que sea la mejor para todas las instancias. En algunos casos las heurísticas sencillas requieren de demasiadas verificaciones para solucionar el problema.

En el apéndice B se puede observar las gráficas respectivas al conjunto de prueba I B.13 y II B.15.

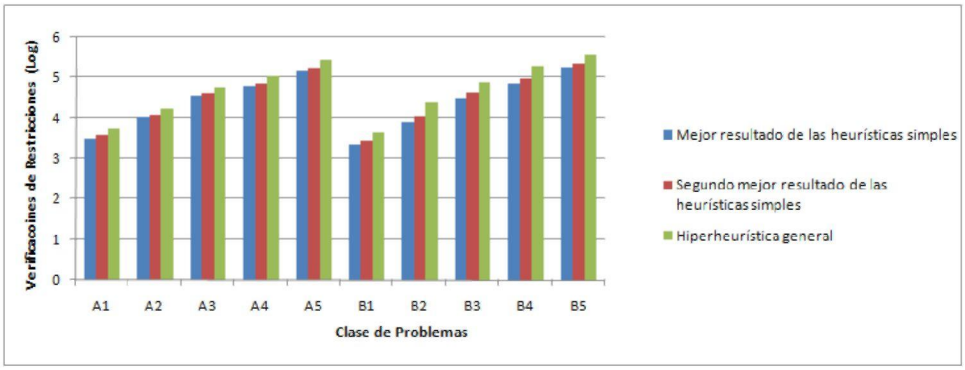

Figura 4.8: Resultados del Experimento III: Conjunto de Entrenamiento

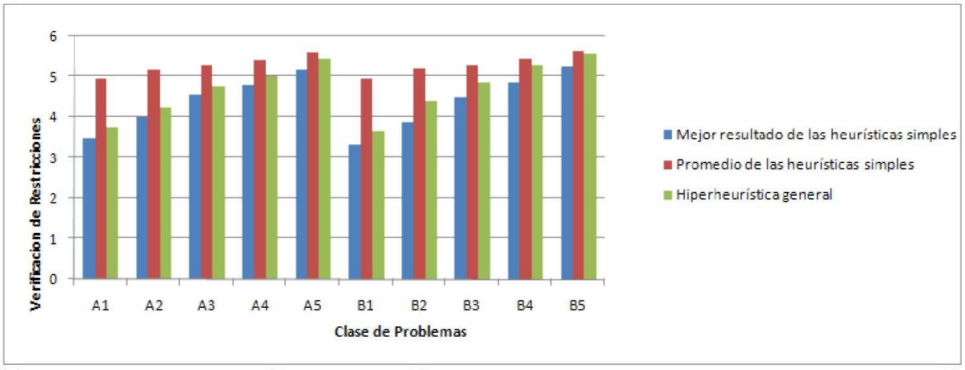

Figura 4.9: Resultados del Experimento III: Conjunto de Entrenamiento, Comparación con el Promedio de las Heurísticas Sencillas

4.3. Discusión General

El primer experimento, la creación de una red de manera empírica sirve para ejemplificar lo difícil que es determinar los parámetros para obtener buenos resultados en la red neuronal. La red expuesta fue la que mejor obtuvo resultados después de una serie de varias pruebas se invirtió un tiempo razonable en la creación de ellas, y aun así no se pudieron crear más de 5 redes neuronales. Es por ello que utilizar el algoritmo genético para la creación de las hiperheurísticas es necesario en este tipo de problemas.

Se obtuvieron tres hiperheurísticas por medio del modelo de solución las cuales fueron evaluadas utilizando los problemas de entrenamiento y problemas de prueba, los cuales son nuevos para la hiperheurística. Las tres hiperheurísticas cuentan con diferentes funciones de evaluación, conservando el mismo número de ciclos y de tamaño de población. En los próximos experimentos se modificará estos parámetros para comparar los resultados. De acuerdo a la literatura la utilización de tres capas es demasiado, por lo que se reducirá a dos en los próximos experimentos. Además se agregará el factor momento.

Aunque el modelo pudiera parecer tardado en generar la hiperheurística; la utilización de la red neuronal no representa un esfuerzo computacional extra al momento de solucionar un CSP, pues el algoritmo feedfoward es lineal. Las hiperheurísticas obtenidas obtuvieron mejoras significativas en un porcentaje pequeño de las instancias, pero la mejora existe, con esto vemos que es factible la creación de las hiperheurísticas, lo que sugiere que se podría mejorar modificando algunos parámetros como el número de ciclos o el tamaño de población.

4.4. Resumen

En esta sección se presentó los resultados preliminares de este proyecto, así como la descripción de las instancias de CPS utilizadas para el mismo. Se observó que las hiperheurísticas no tuvieron un comportamiento competitivo en comparación a la mejor, pero si en comparación al promedio de las heurísticas simples. La existencia de una mejor heurística simple nos hace buscar alternativas para que ésta sea elegida por la hiperheurística al momento de ejecutarse.

Capítulo 5

Experimentos y Resultados Finales

Este capítulo presenta los experimentos y resultados finales producto de esta investigación. Las instancias de CSPs utilizadas se mantienen para ser consistentes con los resultados anteriores.

5.1. Experimentos y Análisis de Resultados

Los conjuntos de entrenamiento y pruebas siguen siendo los mismos que los experimentos preliminares, ahora el objetivo de estos experimentos es determinar una forma más rápida de encontrar las hiperheurísticas manteniendo la calidad de los resultados obtenidos previamente.

En la representación anterior de la red neuronal se consideraban 3 capas ocultas, con un máximo de 15 neuronas por capa. Con lo anterior existe la posibilidad de calcular aproximadamente 27000 pesos para la red ($8 \times 15 \times 15 \times 15 \times 1$). Se propone eliminar una de las capas ocultas con la finalidad de reducir este cálculo de pesos. Al tener solamente dos capas ocultas el número máximo de pesos se reduce a 1800 pesos. Lo que conlleva una reducción del tiempo de entrenamiento.

Otro de los cambios introducidos en este modelo es la utilización del factor *momento*. Evidencia empírica demuestra que el uso del momento en el algoritmo de backpropagation puede ser útil en acelerar la convergencia y evitar mínimos locales. La idea sobre usar momento es estabilizar la actualización de pesos usando una combinación del peso actual con una fracción del peso anterior. Esto le da al sistema cierta cantidad de inercia. El vector de pesos tenderá a la misma dirección. El momento tiene los siguientes efectos:

- Suaviza los cambios y suprime las oscilaciones de lado a lado sobre el error.
- Cuando todos los cambios están en la misma dirección el momento amplifica el rango de aprendizaje causando una convergencia más rápida.
- Permite escapar de pequeños mínimos locales.

A manera de resumen se presenta nuevamente la descripción del cromosoma utilizado mencionando la adición del factor momento, así como la eliminación de una capa oculta. El primer parámetro codificado es el número de neuronas que tendrá la primer capa oculta, esta capa tiene al menos una neurona y quince como máximo. Es importante hacer notar que de esta manera aseguramos la existencia de al menos una capa oculta; este parámetro utiliza cuatro bits para su codificación. Los siguiente parámetro se refiere al número de neuronas en la segunda capa oculta, a diferencia de la anterior, éstas si pueden tomar el valor de cero, ya que de acuerdo a la teoría solo es necesario contar con una capa oculta, y toma su máximo en quince, y de igual manera utiliza cuatro bits para su codificación. Para mejorar el desempeño de la regla de aprendizaje de retropropagación es usual utilizar un término de momentum que tiene el efecto de hacer que el peso se siga moviendo en la dirección en la que se movió en los ciclos anteriores de aprendizaje. El uso del momentum tiene dos efectos. Primero, los pesos tiene una aceleración en las zonas donde el gradiente del error es pequeño. Segundo, los pesos pueden pasar sobre pequeñas lomas en la curva del error debido a la velocidad que gana antes de llegar a cada una. Este parámetro fue codificado en 20 bits tomando su valor mínimo en 0 y valor máximo en 1, el máximo de este valor. Este es el tercer parámetro del cromosoma. Como se ha mencionado, se utiliza el algoritmo de backpropagation para el entrenamiento de la red, por lo que es necesario definir el número de épocas el cual será ejecutado, este parámetro está limitado por un mínimo de 444 y un máximo de 1000, para la codificación de este elemento se utilizó 10 bits, el valor mínimo es el número de registros en el set de entrenamiento. Los parámetros que define el estado de un CSP se encuentran normalizados entre los valores -3 y 3, es por ello que el peso mínimo y máximo de las sinapsis se encuentra también entre estos valores respectivamente, y por último se encuentra codificado el rango de aprendizaje de la red, este valor toma valores entre 0.1 y 0.8, con 20 bits para su codificación.

El esquema del nuevo modelo puede ser visto en la figura 5.1, donde se puede notarse el cambio que se ha mencionado.

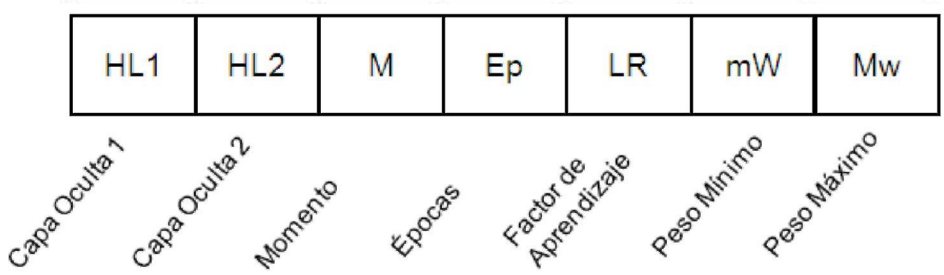

Figura 5.1: Representación del Cromosoma

5.1.1. Experimento IV

Las características de este experimento son las siguientes, se utiliza un algoritmo genético de estado estable con selección de torneo tamaño 2, con una probabilidad de cruce de 1 y una probabilidad de mutación de .01. La función de evaluación es la explicada en el modelo de solución, es decir, el promedio de dividir el número de verificación de restricciones de la heurística sencilla entre el número de verificación de restricciones de la hiperheurística. El número de ciclos es de 100 y el número de individuos de la población de 20. La diferencia con respecto al experimento preliminar III es el cambio de cromosoma. Se presentan los resultados obtenidos en la tabla 5.3. Los parámetros del algoritmo genético utilizado pueden verse en la tabla 5.1

Dato	**Valor**
Población	20 individuos
Ciclos	100
Probabilidad de Cruce	1
Probabilidad de Mutación	0.01
Algoritmo Genético	Estado Estable

Cuadro 5.1: Detalles del algoritmo genético utilizado

La hiperheuristica obtenida es la que se presenta en la tabla 5.2.

Dato	**Valor**
Capa Oculta 1	6 neuronas
Capa Oculta 2	3 neuronas
Momento	0.4409905841585511
Épocas	627.0
Factor de Aprendizaje	0.17020835275867588
Mínimo Peso	−1.9501918945235197
Máximo Peso	0.8299510835139419
Aptitud Obtenida	0.9193954276364712

Cuadro 5.2: Detalles de la Hiperheurística obtenida

La red que se obtiene no presenta diferencia significativa a los resultados previos. En el 94 % de los casos para en conjunto de entrenamiento se requirió de más del 15 % de verificaciones de restricciones. Y en el 1.33 % se reduce en más del 15 %, esto para la clase A. En la clase B los resultados son similares. En el 89.33 % se requiere de más del 15 % de verificaciones y un el 2.33 % las verificaciones se reducen.

Para los casos de Prueba I se observa lo esperado, al no aprender el conjunto de entrenamiento el hiperheurística no compite con la heurística sencilla. En el 95 % de los

		Reducción		Igual	Incremento	
Conjunto	**Clase**	$> 15\,\%$	$15\,\%$ **a** $3\,\%$	$\pm 3\,\%$	$3\,\%$ **a** $15\,\%$	$> 15\,\%$
Entrenamiento	A	1.33	0.67	1.00	3.00	94.00
Entrenamiento	B	2.33	0.67	3.00	4.67	89.33
Prueba I	A	0.50	0.50	2.50	1.50	95.00
Prueba I	B	4.50	1.00	2.00	7.00	85.50
Prueba II	C	1.20	0.80	8.40	4.00	85.60

Cuadro 5.3: Resultados del Experimento IV

casos no logra la mejora para la clase A, mientras que en B en el 4.5 % de las instancias las reducción de más del 15 % si se presenta.

En el conjunto de Prueba II la hiperpehurística sigue siendo mala no logrando la reducción de las verificaciones en el 85.6 % de los casos. Mientras que en el 1.2 % esta reducción fue presente.

5.1.2. Experimento V

Este experimento continúa con los parámetros del experimento anterior con las diferencias siguientes: El número de individuos de la población es de 30 y el número de ciclos del algoritmo genético es de 200. Con esto se espera un cambio en los resultados que se han estado obteniendo.

Los parámetros del algoritmo genético utilizado pueden verse en la tabla 5.4

Dato	**Valor**
Población	30 individuos
Ciclos	200
Probabilidad de Cruce	1
Probabilidad de Mutación	0.01
Algoritmo Genético	Generacional

Cuadro 5.4: Detalles del algoritmo genético utilizado

La hiperheurística obtenida se presenta en la tabla 5.5:

Los resultados obtenidos se presentan en la tabla 5.6

Sorpresivamente en esta ocasión la hiperheurística aprende a comportarse como la mejor heurística sencilla, se recuerda que la heurística sencilla no siempre es la misma en todas las instancias. En el conjunto de entrenamiento para la clase A la hiperheurística logra comportarse como la mejor heurística simple en el 69.67 % de las instancias. En la clase B el comportamiento fue similar, en el 75.33 % de los CSPs se obtiene los

Dato	Valor
Capa Oculta 1	4 neuronas
Capa Oculta 2	1 neurona
Momento	0.27183530434343856
Épocas	840
Factor de aprendizaje	0.7334886063397021
Mínimo Peso	−0.41416071269932075
Máximo Peso	−0.1970086622633902
Aptitud Obtenida	0.9528254007664921

Cuadro 5.5: Detalles de la Hiperheurística obtenida

		Reducción		**Igual**	**Incremento**	
Conjunto	**Clase**	$> 15\,\%$	$15\,\%$ **a** $3\,\%$	$\pm 3\,\%$	$3\,\%$ **a** $15\,\%$	$> 15\,\%$
Entrenamiento	A	0.00	0.00	69.67	13.33	17.00
Entrenamiento	B	0.00	0.00	75.33	5.00	19.67
Prueba I	A	0.00	0.00	75.50	10.00	14.50
Prueba I	B	9.00	0.50	54.50	10.00	26.00
Prueba II	C	0.00	0.00	71.60	7.20	21.20

Cuadro 5.6: Resultados del Experimento V

mismos resultados. Para este conjunto, el conjunto de entrenamiento, no hubo mejora significativa, es decir, no hubo reducción del número de verificaciones de restricciones.

En el conjunto de prueba I en la clase A, la hiperheurística se comporta como la mejor heurística simple en el 75.50 % de los casos, mientras que en los problemas de la clase B en el 54.50 %; pero hay que notar que hubo una mejora significativa en el 9 % de las instancias. Para el cojunto de prueba II, la hiperheurística se comportó como la mejor hiperheurística en el 71.6 % de las instancias.

La gráfica 5.2 muestra que la hiperheurística tiene el mismo comportamiento que la heurística sencilla, puesto una está sobre la otra. Y en comparación con la segunda mejor heurística es mejor que ella.

En comparación con el promedio la hiperheurística tiene un comportamiento competitivo, esto se aprecia en la gráfica 5.3.

Las gráficas para los conjuntos de prueba se encuentran en el apéndice C

5.1.3. Experimento VI

En este experimento modificamos el número de ciclos corriéndolo por 250 ciclos, puesto que uno de los experimentos corrió por 200 ciclos y otro por 300 ciclos. Se mantuvo la siguiente función de evaluación: La población inicial es evaluada en primera

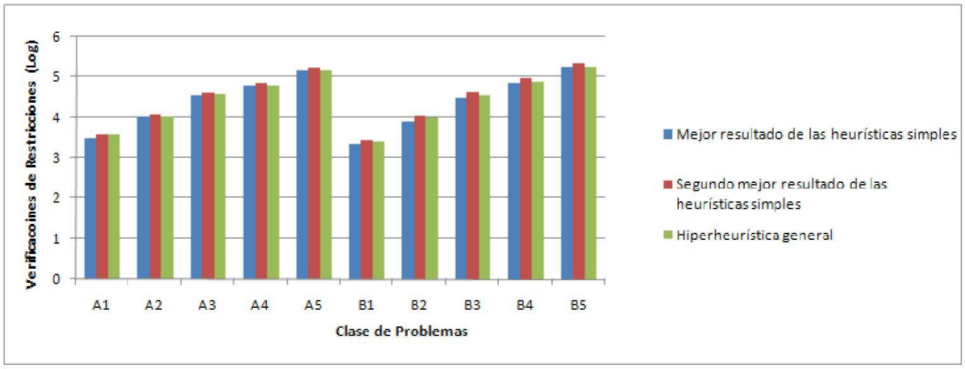

Figura 5.2: Resultados del Experimento V: Conjunto de Entrenamiento

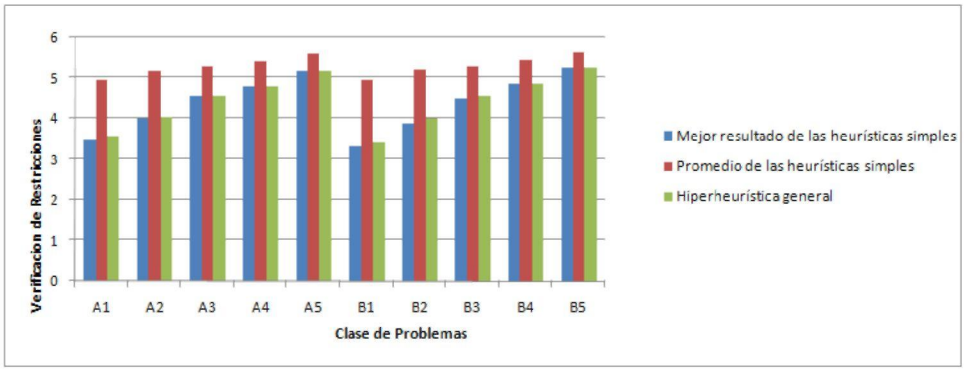

Figura 5.3: Resultados del Experimento V: Conjunto de Entrenamiento

instancia con 5 problemas y en cada ciclo se evalúa con un problema más, los nuevos individuos son evaluados con esos 5 problemas y el número de ciclos que la población lleve. Las redes neuronales creadas cuentan con 2 capas y el factor momento. El número de individuos de la población es de 25.

Los parámetros del algoritmo genético utilizado pueden verse en la tabla 5.7

Dato	**Valor**
Población	25 individuos
Ciclos	250
Probabilidad de Cruce	1
Probabilidad de Mutación	0.01
Algoritmo Genético	Generacional

Cuadro 5.7: Detalles del algoritmo genético utilizado

La hiperheurística obtenida se encuentra descrita en la tabla 5.8.

Los resultados obtenidos se presentan en la tabla 5.9.

Esta hiperheurística aprende el set de entrenamiento. Tanto para la clase A y B

Dato	Valor
Capa Oculta 1	5 neuronas
Capa Oculta 2	8 neuronas
Momento	0.4455745479339761
Épocas	1072.0
Learning Rate	0.3963265358189695
Mínimo Peso	−2.9751555359354835
Máximo Peso	1.8852661145811416
Aptitud Obtenida	0.9674016765241307

Cuadro 5.8: Detalles de la Hiperheurística obtenida

		Reducción		**Igual**	**Incremento**	
Conjunto	**Clase**	$> 15\,\%$	$15\,\%$ **a** $3\,\%$	$\pm 3\,\%$	$3\,\%$ **a** $15\,\%$	$> 15\,\%$
Entrenamiento	A	0.00	0.00	69.67	13.33	17.00
Entrenamiento	B	0.00	0.00	75.33	5.00	19.67
Prueba I	A	0.00	0.00	75.50	10.00	14.50
Prueba I	B	9.00	0.50	54.50	10.00	26.00
Prueba II	C	0.00	0.00	71.60	7.20	21.20

Cuadro 5.9: Resultados del Experimento VI

se comporta como la mejor heurística simple, esto sucede en el 69.67 % de los casos de la clase A y en el 75.33 % de la clase B. No hubo mejora significativa, pero el aprendizaje se da. Al momento de probar los resultados con instancias nuevas para la hiperheurística obtenemos resultados similares a los anteriores, en el 75 % de las instancias del conjunto de prueba I de la clase A la hiperheurística se comportó como la mejor heurística sencilla. En el caso de las instancias de clase B la hiperheurística se comporta como la mejor heurística en el 54.5 % de los casos además de mostrar una mejora significativa en el 9 % de los problemas. Cuando se le presentaron los problemas más difíciles de la clase C del conjunto de prueba II la hiperheurística se comportó como la mejor heurística sencilla en el 71.6 % de los casos.

Un poco más a detalle se puede observar el comportamiento de la hiperheurística en cada subconjunto de la clase a la que pertenece. Los resultados son expuestos en la figura 5.4.

En comparación con el promedio la hiperheurística tiene un comportamiento competitivo, esto se aprecia en la gráfica 5.5.

Las gráficas de los conjuntos de Prueba I y II se encuentra en el apéndice C

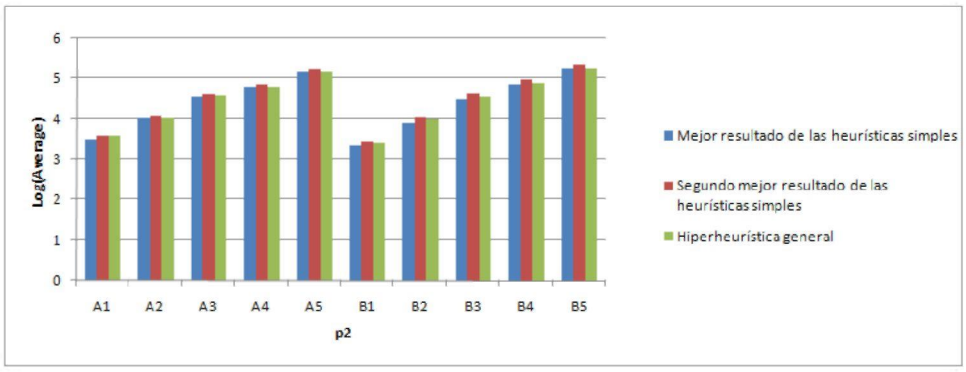

Figura 5.4: Resultados del Experimento IV: Conjunto de Entrenamiento

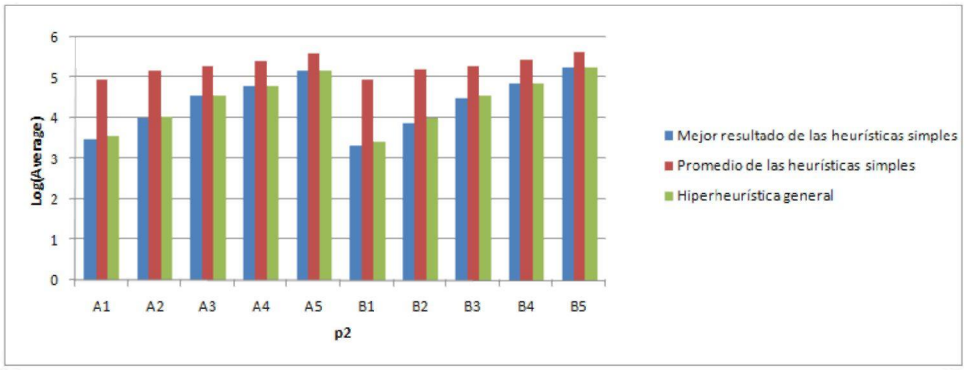

Figura 5.5: Resultados del Experimento VI: Conjunto de Entrenamiento

5.2. Comparación entre la Hiperheurística Obtenida con el Modelo Neuro-Evolutivo y la Hiperheurística Obtenida con el Modelo Evolutivo

Para verificar la generalidad de la hiperheurística obtenida, se realizó una comparación con respecto a una hiperheurística utilizada para crear el conjunto de entrenamiento y la heurística general creada con el modelo obtenido.

La hiperheurística generada con el modelo evolutivo (HHE) es la siguiente: Se ejecutó el modelo por 100 generaciones con 20 individuos en la población. Una probabilidad de cruce de 1 y una probabilidad de mutación de 0.1. Esta hiperheurística obtuvo una aptitud de 1.063390. En la tabla 5.10 se presenta detallada la hiperheurística obtenida.

Los resultados que se obtuvieron con esta hiperheurísta evolutiva son los que se aprecian en la tabla 5.11

Esta hiperheurística fue obtenida de acuerdo a las especificaciones presentadas por Terashima [18], se puede observar que los resultados de la hiperheurística evolutiva son similares a los obtenidos por la hiperheurística neuro-evolutiva. Para el conjunto

v	d	c	l_c	m_c	s_c	r	k	h
0.893	-0.055	-0.388	-1.299	0.867	-0.923	-1.943	-1.491	2
-0.736	-1.6259	1.0069	0.8169	2.054	-1.2649	-0.662	-0.994	1
-0.321	-1.085	-0.622	-2.4089	-0.58	-1.165	1.385	1.27	5
-0.539	-1.185	-0.852	0.58	-1.045	-1.2	-1.089	0.946	0

Cuadro 5.10: Hipherheurística creada con el modelo evolutivo

		Reducción		**Igual**	**Incremento**	
Conjunto	**Clase**	$> 15\,\%$	$15\,\%$ **a** $3\,\%$	$\pm 3\,\%$	$3\,\%$ **a** $15\,\%$	$> 15\,\%$
Entrenamiento	A	0.67	2.33	40.67	29.00	27.33
Entrenamiento	B	2.00	1.00	11.33	21.33	64.33
Prueba I	A	0.50	7.00	38.50	25.00	29.00
Prueba I	B	10.50	1.50	7.00	15.50	65.50
Prueba II	C	2.80	2.80	38.80	17.60	38.00

Cuadro 5.11: Resultados de la Hiperheurística Evolutiva

de entrenamiento en los problemas de clase A la hiperheurística evolutiva se comporta como la mejor heurística simple en el 40.67 % de los casos. y en un 27.33 % se requiere de un número extra de verificaciones de restricciones. Para la clase B del mismo conjunto de instancias se comporta como la mejor heurística sencilla en el 21.33 % de los casos, mientras que en un 64.33 % de las instancias se requiere de más de un 15 % de verificaciones.

Para el conjunto de prueba I, en los problemas de la clase A hubo una mejora significativa en el 0.5 % de los casos, se comporta como la mejor heurística sencilla en el 38.5 % de ellos; mientras que en un 29 % se requiere de un 15 % extra de verificaciones de restricciones. Para las clase B hay una mejora significativa en el 10.50 % de los casos, y se comporta igual que la heuristica sencilla en el 7 % de las instancias. Sin embargo en un 65.50 % se requiere de más de un 15 % extra en verificaciones de restricciones.

En el conjunto de prueba II, la hiperheurística evolutiva se comporta igual que la heurística sencilla en el 38.80 % de las instancias. Mientras que se requiere de un 15 % extra de verificaciones de restricciones en el 38 % de las instancias.

Haciendo la comparación entre la hiperheurística neuro-evolutiva y la hiperheurística evolutiva se presentan los resultados en la tabla 5.12

La hiperheurística neuro-evolutiva se comporta mejor que la hiperheurística evolutiva como se puede apreciar en la tabla 5.12 en ella vemos que las instancias de la clase A del conjunto de entrenamiento existe una reducción significativa en el 16 % de ellas y una reducción considerable en el 26 % de las mismas. Se observa también que se comportan igual en el 51.67 % de las instancias mientras que solo en un pequeño porcentaje 2.67 % se requiere de más del 15 % extra de verificaciones de restricciones. En el

		Reducción		Igual	Incremento	
Conjunto	**Clase**	$> 15\,\%$	$15\,\%$ **a** $3\,\%$	$\pm 3\,\%$	$3\,\%$ **a** $15\,\%$	$> 15\,\%$
Entrenamiento	A	16.00	26.00	51.67	3.67	2.67
Entrenamiento	B	56.67	23.33	9.67	2.00	8.33
Prueba I	A	20.00	26.00	44.00	8.00	2.00
Prueba I	B	63.00	12.00	7.00	3.00	15.00
Prueba II	C	26.40	16.00	47.60	5.20	4.80

Cuadro 5.12: Comparativa entre las Hiperheurística Evolutiva y Neuro-Evolutiva

caso de la clase B las mejoras significativas se incrementaron al 56.67 % y la reducción considerable en el 23.33 % y ya con estos dos rangos la mayoría de los problemas han sido solucionados. Dejando un pequeño porcentaje para las verificaciones extras.

En el conjunto de prueba I, la hiperheurística neuro-evolutiva sigue comportándose mejor la que hiperheurística evolutiva. En las instancias de la clase A se logra una reducción de más del 15 % de las verificaciones de restricciones en un 20 % de las instancias, en un 26 % de las instancias se logra una reducción entre el 3 % y 15 % de las verificaciones de restricciones. En el 44 % de las instancias se logra que ambas hiperheurísticas se comporten de igual manera.

El conjunto de prueba II, el comportamiento continua igual , la hiperheurística neuro-evolutiva se comporta mejor la que hiperheurística evolutiva. En un 26.40 % de las instancias la reducción fue significativa, es decir, más del 15 % menos de verificaciones de restricciones. En un 16 % de las instancias se logra la reducción entre el 3 % y el 15 % de las verificaciones. En el 47.60 % de las instancias ambas hiperheurísticas se comportan de manera similar.

Estas mejoras se aprecian más en las gráficas presentadas. En la gráfica 5.6 se observa como la hiperheurística neuro-evolutiva es mejor que la hiperheurística evolutiva. Por ejemplo en los problemas de la clase B1 y B2 del conjunto de entrenamiento la reducción es notoria. Hay que notar también que la hiperheurística neuroevolutiva se comporta en la mayoría de las instancias como la mejor heurística sencilla.

En en apéndice C se puede observar las gráficas del conjunto de prueba I y conjunto de prueba II.

5.3. Discusión General

Los experimentos aquí presentados requieren de un tiempo de cómputo demasiado extenso llegando a requerir hasta 4 días de procesamiento, pero una vez obtenida la hiperheurística esta puede ser aplicada de manera rápida en la solución de las instancias de los CSPs.

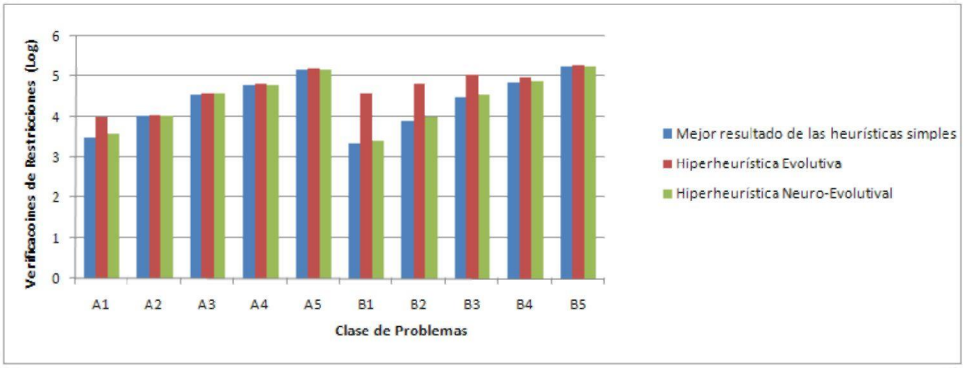

Figura 5.6: Comparación entre hiperheurística neuro-evolutiva y evolutiva: Conjunto de Entrenamiento

Analicemos un poco los resultados presentados. En el capitulo anterior las hiperheurísticas obtenidas presentan un comportamiento nada competitivo contra la aplicación de una heurística sencilla; pero hay que notar que las hiperheurísticas, aunque en la mayoría de las instancias requería de un mayor número de verificaciones, existía un pequeño porcentaje de mejoras significativas. Este pequeño porcentaje da la idea de que algún parámetro del experimento podría ser modificado para lograr que se incrementara las instancias resueltas de manera más eficiente.

Teniendo en mente lo anterior se diseñaron los experimentos presentados en este capítulo, lo primero que se hizo fue modificar la función de evaluación, reducir el número de capas ocultas de 3 a 2 y agregando el termino momento. El experimento fue realizando manteniendo los mismos parámetros de tamaño de población y número de ciclos. Los resultados ya presentados son similares a los anteriores, un gran porcentaje requiere de verificaciones de restricción extras pero aun están presentes las instancias donde la mejora es significativa.

Se había modificado ciertos parámetros del cromosoma, ahora se modificaría el número de individuos en la población y el número de ciclos del AG. Los resultados dieron un giro inesperado. Ahora un gran porcentaje de las instancias eran solucionadas como la mejor heurística simple, es decir, la red neuronal utilizada como hiperheurística esta bien entrenada con la tabla de entrenamiento, y puede generalizar los resultados a problemas nuevos; pero las pocas mejoras significativas que habían desaparecieron. Ahora la hiperheurística es capaz de elegir entre las heurísticas sencillas aquella que nos dará el menor número de verificaciones, pero no reduce el número de ellas.

5.4. Resumen

En este capítulo se presentaron los experimentos y resultados que permiten demostrar las fortalezas y debilidades del modelo utilizado. Las hiperheurísticas generadas

por el modelo neuro-evolutivo presentado producen resultados competitivos, que mejoran el desempeño de la mejor heurística de bajo aplicada. Se presenta también la comparación entre las hiperheurísticas obtenidas con los dos cromosomas presentados en este proyecto de investigación.

Capítulo 6

Conclusiones

En este capítulo se presentan las conclusiones y contribuciones derivadas de esta investigación, así como también se presentan algunas sugerencias para extender el trabajo presentado en este documento.

6.1. Conclusiones

Este proyecto de investigación tiene como hipótesis principal la factibilidad de crear hiperheurísticas para el ordenamiento dinámico de variables en CSPs utilizando un enfoque neuro-evolutivo, y de acuerdo a lo observado en los experimentos se demostró que la hipótesis es verdadera. Para la generación de estas hiperheurísticas se utiliza un algoritmo genético de longitud constante en el cual se codifican dos elementos principales de la red neuronal: la arquitectura de la misma y algunos parámetros del algoritmo de entrenamiento (en este caso backpropagation). La entrada de la red neuronal es la representación del estado del problema y la salida de la misma es la heurística a aplicar en un momento dado de la solución. El proceso neuro-evolutivo requiere de dos procesos de entrenamiento, el proceso propio del algoritmo genético y de la red neuronal, lo que hace que se requiera mucho tiempo computacional; pero la red neuronal ya entrenada, es decir, la hiperheurística, obtiene la respuesta de manera rápida. Las hiperheurísticas obtenidas fueron utilizadas para resolver cada una de las instancias presentadas en este trabajo.

Las hiperheurísticas obtenidas con este modelo de acuerdo a los resultados logran comportarse mejor que el promedio de las heurísticas simples; y en comparación a la mejor heurística sencilla se comporta como ella en la mayoría de los casos. Se podría pensar que no existe motivo para crear una hiperheurística que toma varios días para ser entrenada si al final se obtendría los mismo resultados que la mejor heurística sencilla. Pero el motivo si es válido ya que la hiperheurística se comporta como la mejor cualquiera que esta sea, es decir, cuando se tiene un problema no sabemos cuál es la mejor heurística a aplicar, y con este modelo ya no es necesario saber cual es puesto que se comporta como tal.

6.2. Contribuciones

Esta investigación contribuye a enriquecer el conocimiento acerca de los modelos de generación de hiperheurísticas para el ordenamiento dinámico de variables en CSPs. Además, presenta una comparación clara entre el desempeño del mejor resultado de las heurísticas de bajo nivel y las hiperheurísticas generadas mediante el modelo. De forma más específica, las contribuciones de este trabajo son:

- Diseñar e implementar una herramienta para la generación de hiperheurísticas generales para ordenar dinámicamente variables en CSPs.
- Comprobar la efectividad del modelo basado en redes neuronales para la generación de hiperheurísticas generales para ordenar dinámicamente variables en CSPs.
- Desarrollar una función de evaluación adecuada para el modelo en su fase de entrenamiento.
- Presentar información acerca de los resultados derivados del uso de hiperheurísticas para el ordenamiento dinámico de variables en diversas instancias de CSPs.

6.3. Trabajo Futuro

A continuación se presentan algunas ideas sobre las cuales se podría extender el trabajo realizado en esta investigación:

- **Utilizar genes reales en lugar de genes binarios**: Los parámetros utilizados para representar la red neuronal toman valores reales por lo que una representación real podría dar ciertas ventajas con respecto a los binarios.
- **Buscar una nueva forma para codificar la red neuronal**: La representación de la red neuronal podría contener más parámetros, se podrían buscar el entrenamiento de la red por medio del algoritmo genético y no por medio del algoritmo de retropropagación del error, otro cambio respecto a la codificación de la red es determinar las neuronas que están interconectadas dentro de la red.
- **Probar con una red neuronal con múltiples neuronas en la capa de salida**: De acuerdo con investigaciones las redes neuronales que tienen múltiples neuronas de salida tienen mejores resultados que las redes de una sola capa.
- **Utilizar otro tipo de redes neuronales**: Realizar experimentos con redes LVQ y Kohonen para comprobar como se comporta el modelo.

- Realizar experimentos con redes neuronales donde no sea necesario una tabla de entrenamiento, es decir, utilizar redes neuronales sin que éstas requieran de otro proyecto. Ya que este modelo fue la generalización de un proyecto previo.

Apéndice A

Distribución de las Heurísticas Sencillas en los Problemas

Los conjuntos de Entrenamiento, Prueba I y Prueba II fueron solucionados utilzando todas las heurísticas sencillas. Una vez solucionadas se elige la mejor heurística sencilla y es la utilizada para comparar con el resultado obtenido por la heurística sencilla.

En la tabla A.1, vemos el comportamiento de las heurísticas en los diferentes cojuntos y en la gráfica A.1 se observa lo presentado en la tabla.

Heuristica	**Entrenamiento**	**Prueba I**	**Prueba II**
FF	38	19	32
Rho	28	18	12
Kappa	403	283	157
E(N)	122	72	45
Bz	0	0	0
FFx	9	8	4
Min-C	0	0	0

Cuadro A.1: Heurísticas Sencillas en los diferentes conjuntos

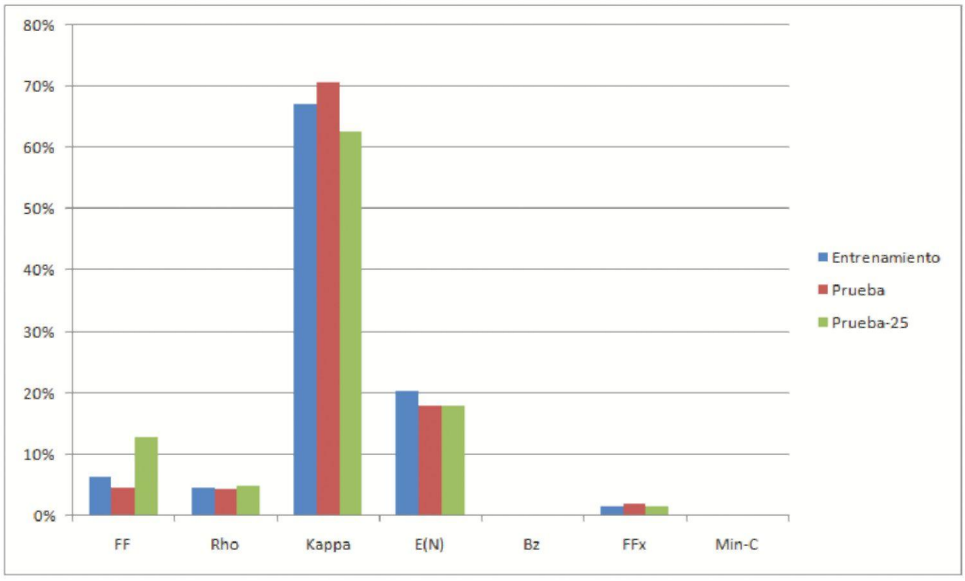

Figura A.1: Distribución de las Heurísticas

Apéndice B

Gráficas del Capítulo 4

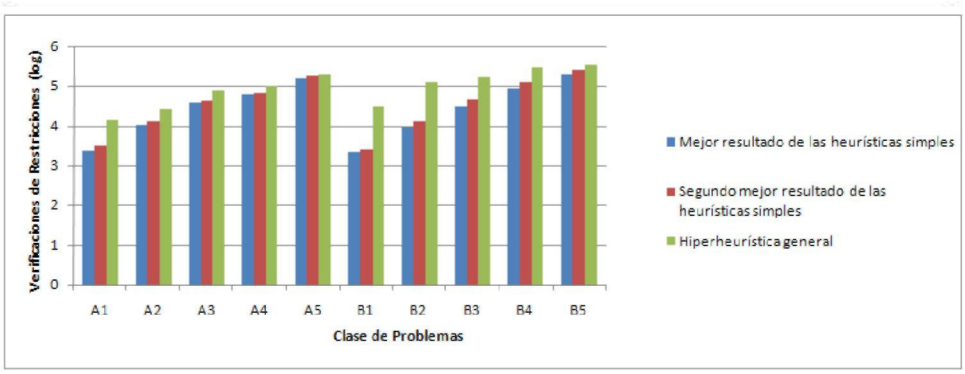

Figura B.1: Resultados del Experimento RN Empírica: Conjunto de Prueba I

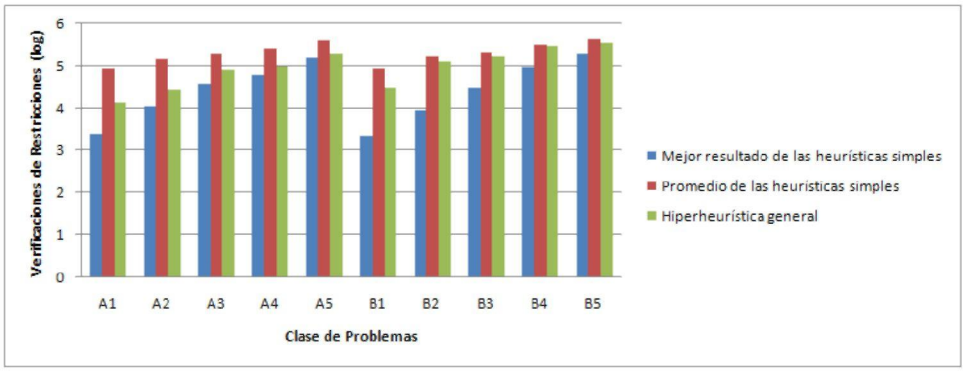

Figura B.2: Resultados del Experimento RN Empírica: Conjunto de Prueba I, Comparación con el Promedio de las Heurísticas Sencillas

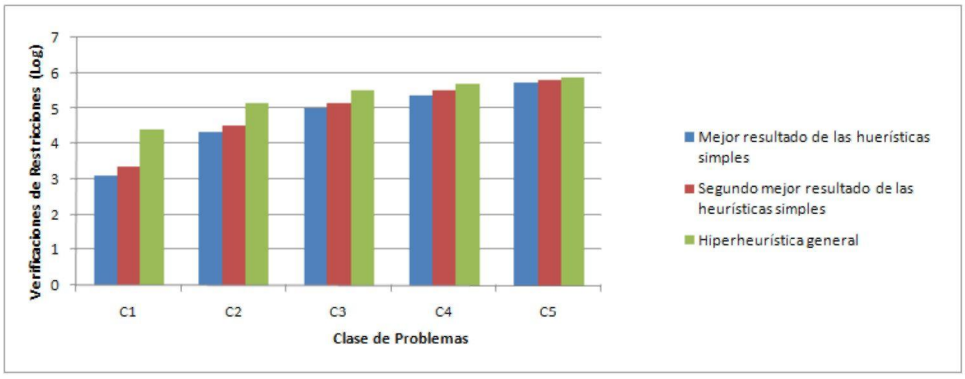

Figura B.3: Resultados del Experimento RN Empírica: Conjunto de Prueba II

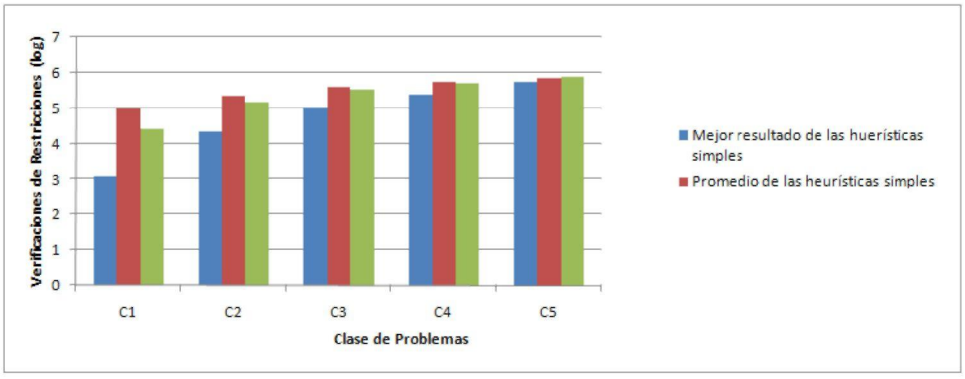

Figura B.4: Resultados del Experimento RN Empírica: Conjunto de Prueba II, Comparación con el Promedio de las Heurísticas Sencillas

Figura B.5: Resultados del Experimento I: Conjunto de Prueba I

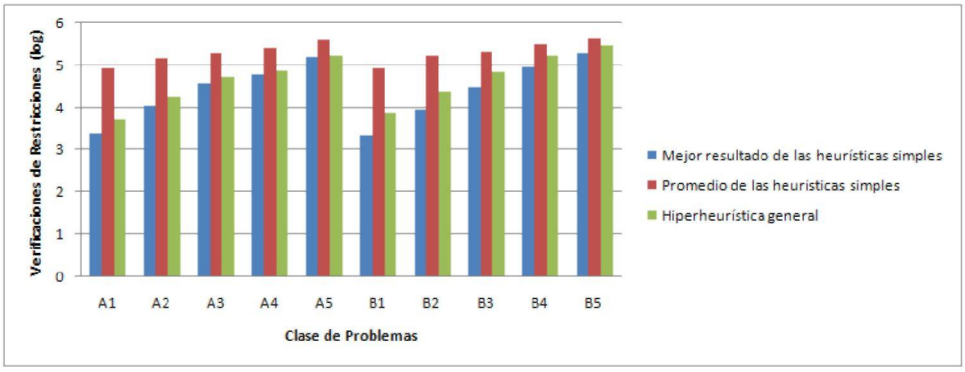

Figura B.6: Resultados del Experimento I: Conjunto de Prueba I, Comparación con el Promedio de las Heurísticas Sencillas

Figura B.7: Resultados del Experimento I: Conjunto de Prueba II

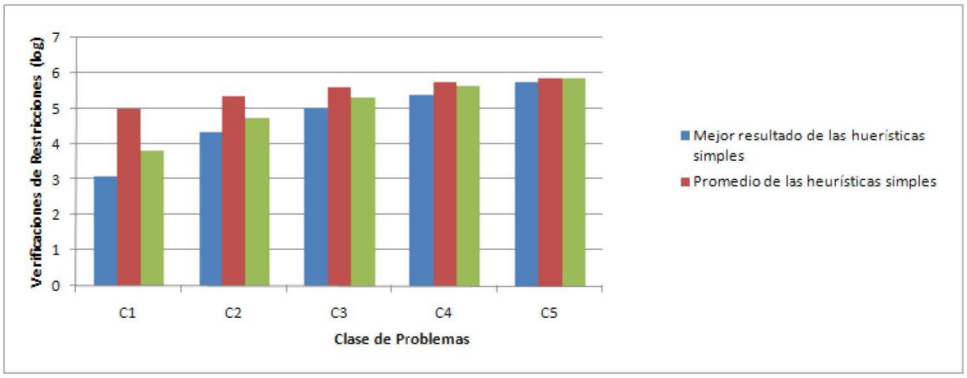

Figura B.8: Resultados del Experimento I: Conjunto de Prueba II, Comparación con el Promedio de las Heurísticas Sencillas

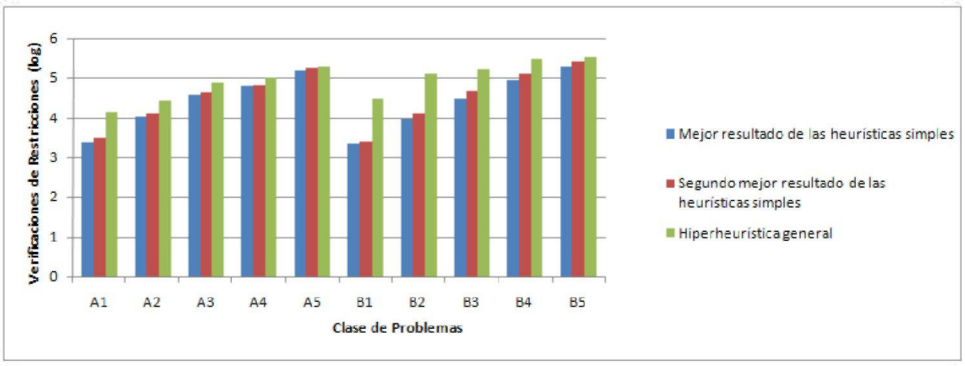

Figura B.9: Resultados del Experimento II: Conjunto de Prueba I

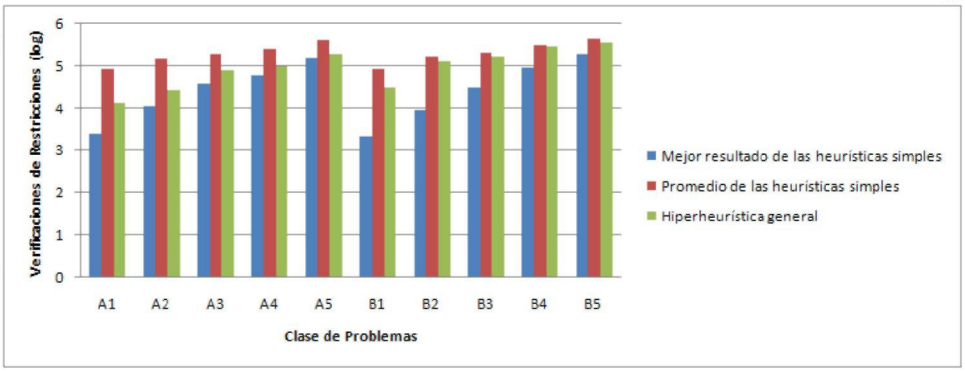

Figura B.10: Resultados del Experimento II: Conjunto de Prueba I, Comparación con el Promedio de las Heurísticas Sencillas

Figura B.11: Resultados del Experimento II: Conjunto de Prueba II

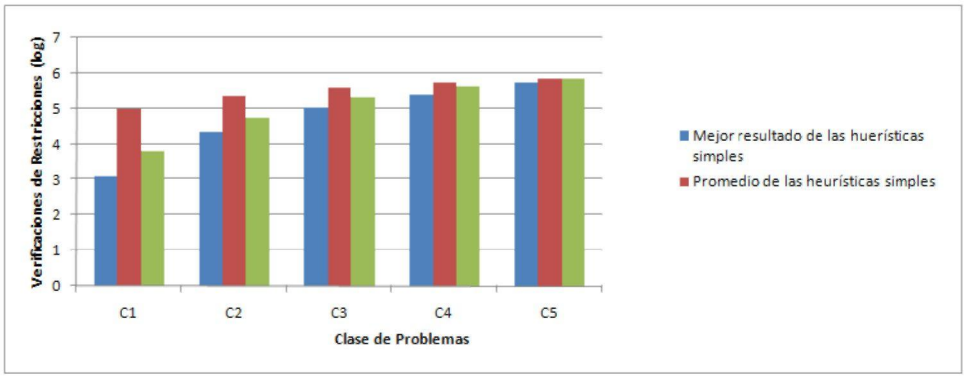

Figura B.12: Resultados del Experimento II: Conjunto de Prueba II, Comparación con el Promedio de las Heurísticas Sencillas

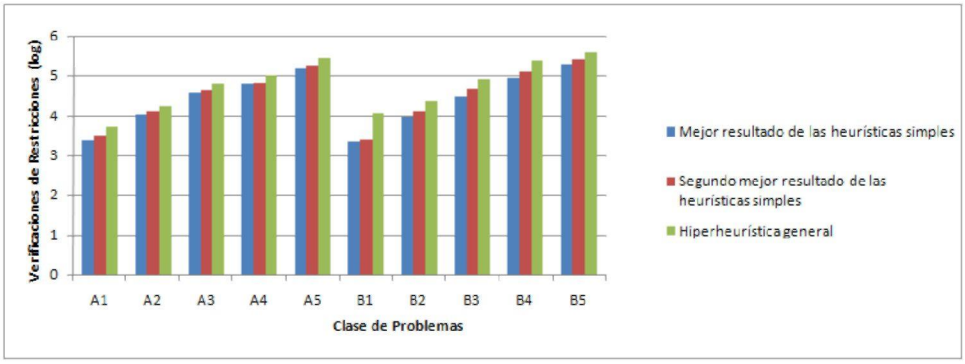

Figura B.13: Resultados del Experimento III: Conjunto de Prueba I

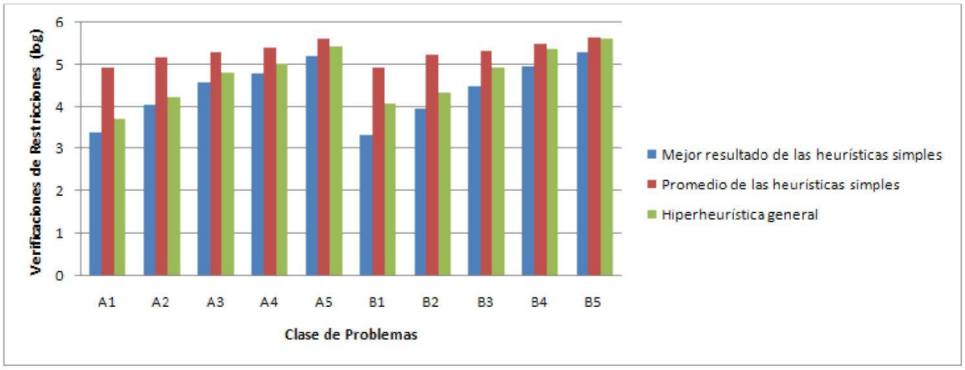

Figura B.14: Resultados del Experimento III: Conjunto de Prueba I, Comparación con el Promedio de las Heurísticas Sencillas

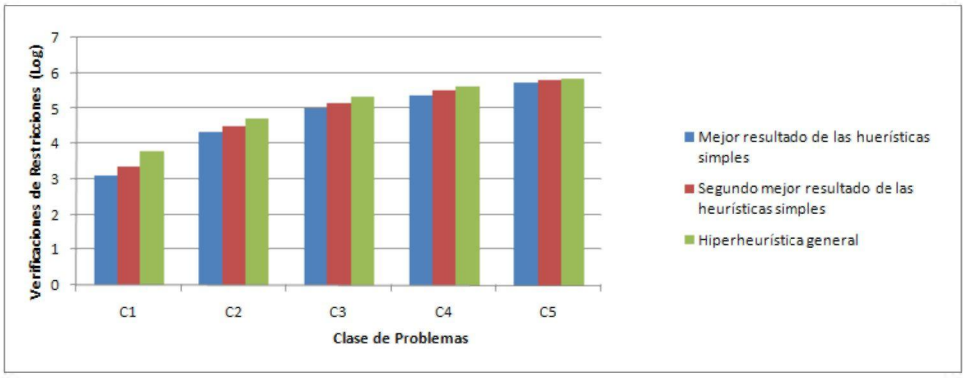

Figura B.15: Resultados del Experimento III: Conjunto de Prueba II

Figura B.16: Resultados del Experimento III: Conjunto de Prueba II, Comparación con el Promedio de las Heurísticas Sencillas

Apéndice C

Gráficas del Capítulo 5

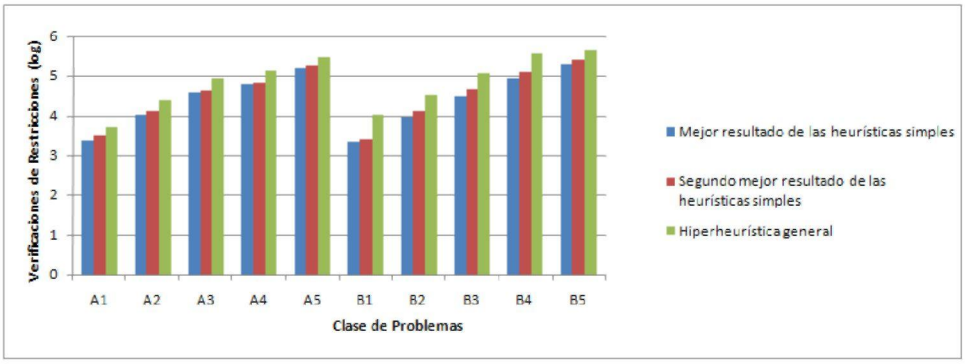

Figura C.1: Resultados del Experimento IV: Conjunto de Prueba I

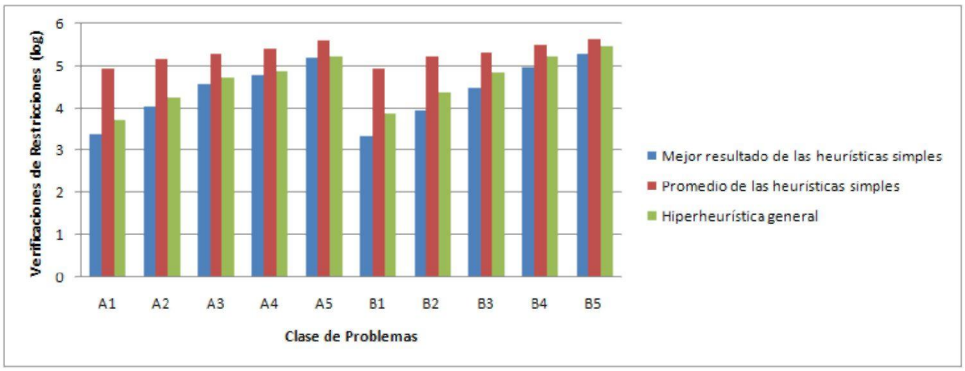

Figura C.2: Resultados del Experimento IV: Conjunto de Prueba I, Comparación con el Promedio de las Heurísticas Sencillas

Figura C.3: Resultados del Experimento IV: Conjunto de Prueba II

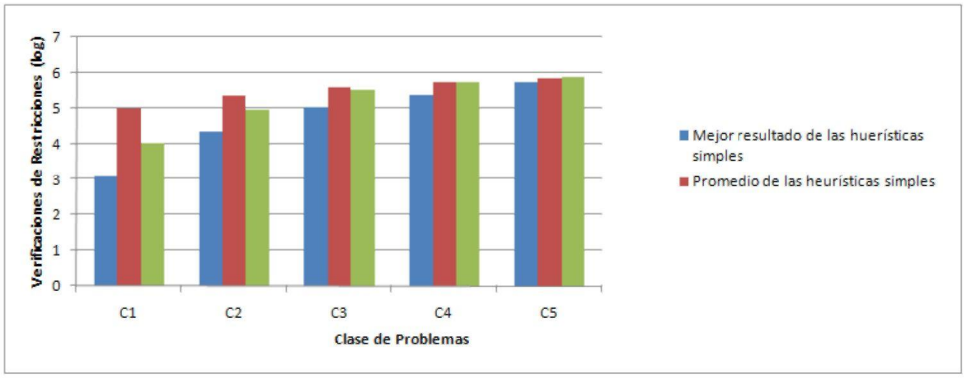

Figura C.4: Resultados del Experimento IV: Conjunto de Prueba II, Comparación con el Promedio de las Heurísticas Sencillas

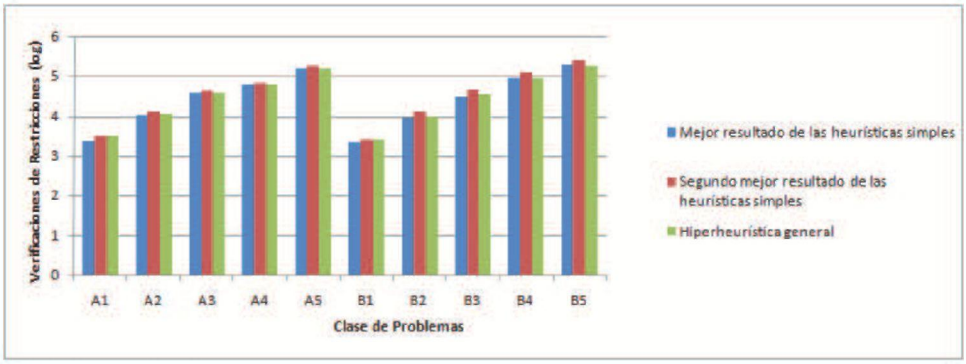

Figura C.5: Resultados del Experimento V: Conjunto de Prueba I

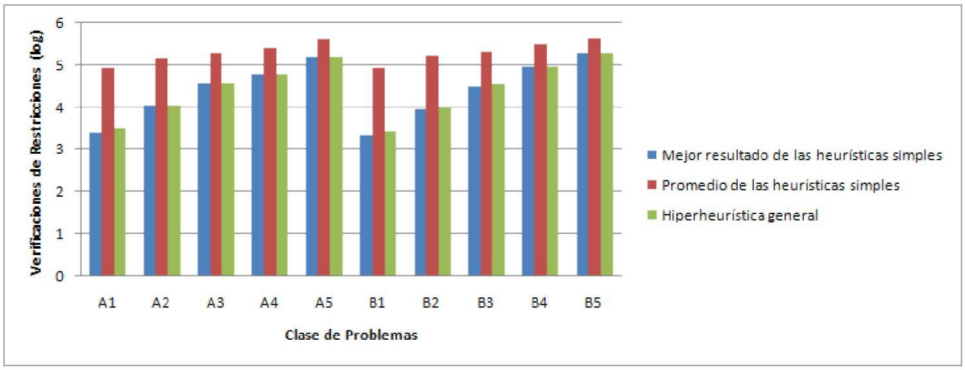

Figura C.6: Resultados del Experimento V: Conjunto de Prueba I, Comparación con el Promedio de las Heurísticas Sencillas

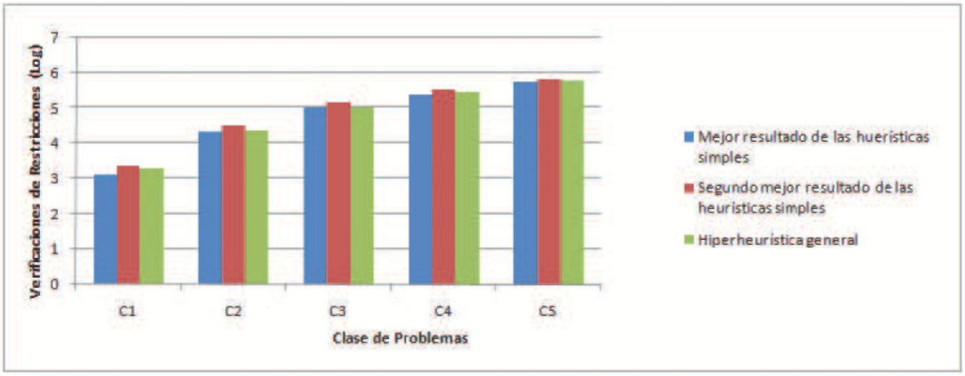

Figura C.7: Resultados del Experimento V: Conjunto de Prueba II

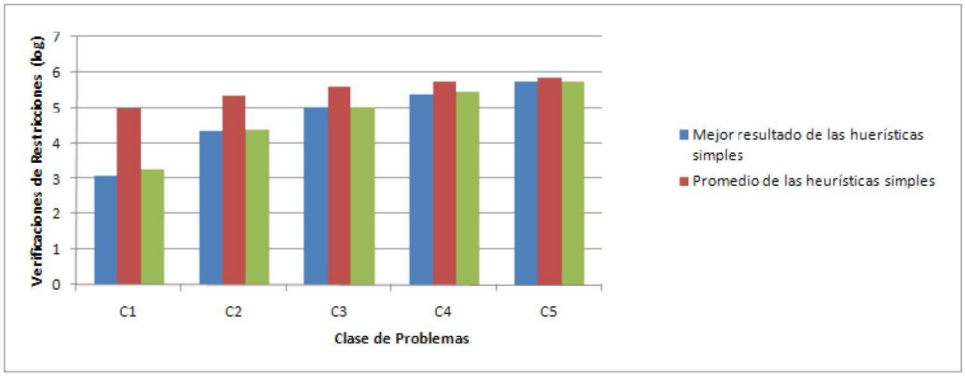

Figura C.8: Resultados del Experimento V: Conjunto de Prueba II, Comparación con el Promedio de las Heurísticas Sencillas

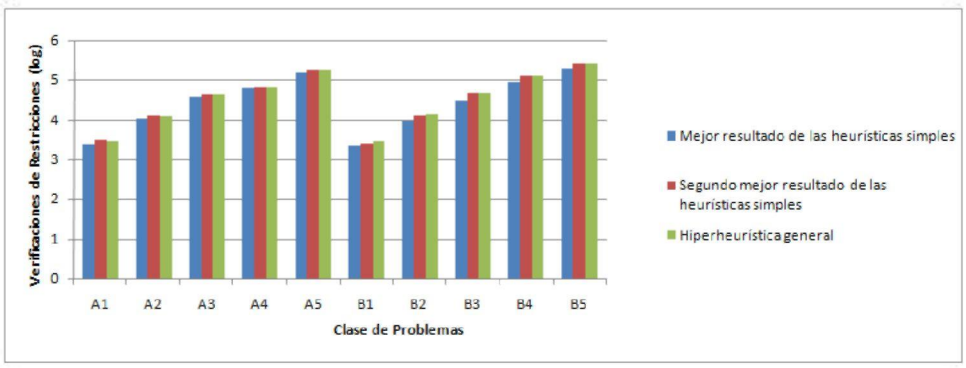

Figura C.9: Resultados del Experimento VI: Conjunto de Prueba I

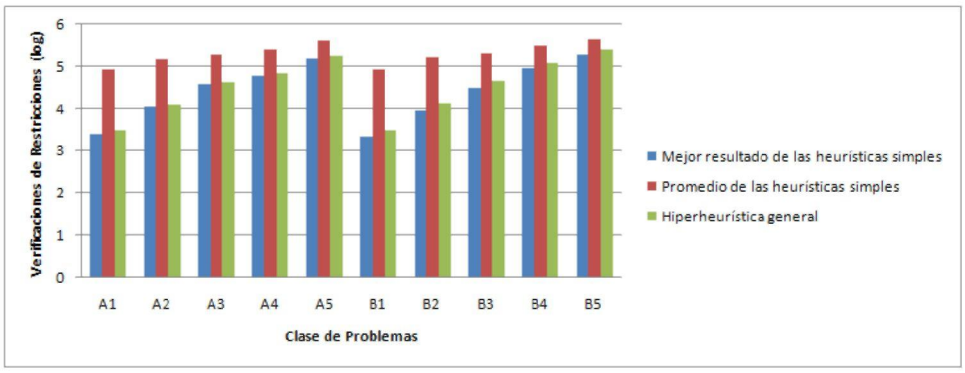

Figura C.10: Resultados del Experimento VI: Conjunto de Prueba I, Comparación con el Promedio de las Heurísticas Sencillas

Figura C.11: Resultados del Experimento VI: Conjunto de Prueba II

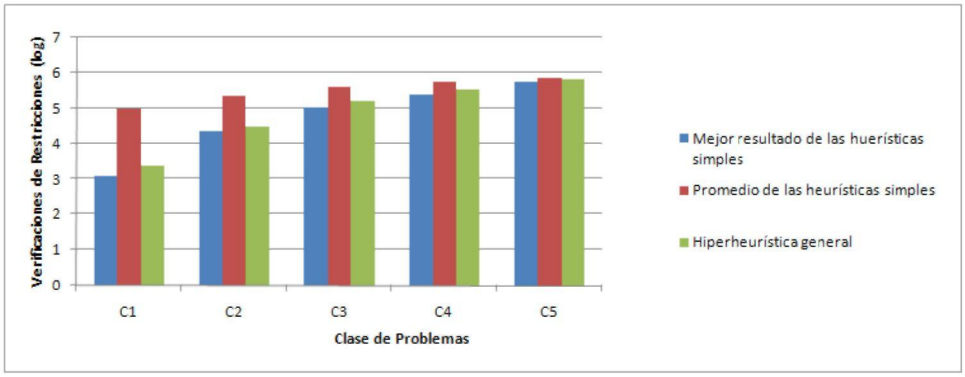

Figura C.12: Resultados del Experimento VI: Conjunto de Prueba II, Comparación con el Promedio de las Heurísticas Sencillas

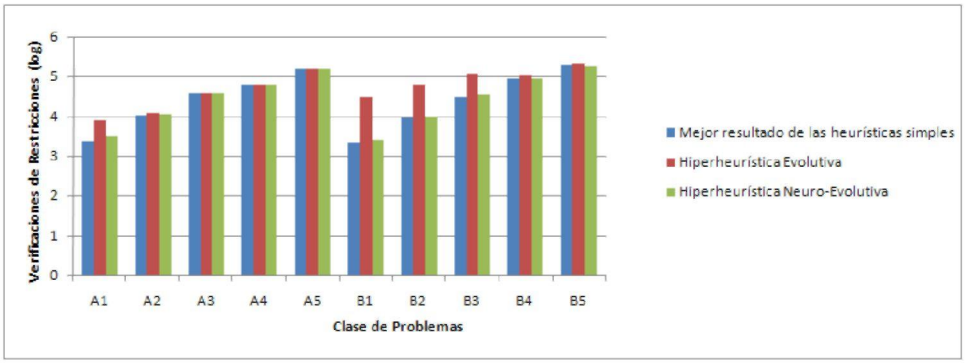

Figura C.13: Comparación entre hiperheurística neuro-evolutiva y evolutiva: Conjunto de Prueba I

Figura C.14: Comparación entre hiperheurística neuro-evolutiva y evolutiva: Conjunto de Prueba II

Bibliografía

[1] F. Bacchus and P. van Run. Dynamic variable ordering in csps. In *CP '95: Proceedings of the First International Conference on Principles and Practice of Constraint Programming*, pages 258–275, London, UK, 1995. Springer-Verlag.

[2] E. Burke, G. Kendall, J. Newall, E. Hart, P. Ross, and S. Schulenburg. Hyper-heuristics: An emerging direction in modern search technology. pages 457–474. 2003.

[3] K. Dowsland, E. Soubeiga, and E. Burke. Solving a shipper rationalisation problem using a simulated-annealing based hyperheuristic. (NOTTCS-TR-2004-1), 2004.

[4] I. P. Gent, E. MacIntyre, P. Prosser, B. M. Smith, and T. Walsh. An empirical study of dynamic variable ordering heuristics for the constraint satisfaction problem. *Principles and Practice of Constraint Programming*, pages 179–193, 1996.

[5] I. P. Gent, E. MacIntyre, P. Prosser, B. M. Smith, and T. Walsh. An empirical study of dynamic variable ordering heuristics for the constraint satisfaction problems. Technical report, University of Leeds School for Computer Studies, 1996.

[6] D. E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley Professional, January 1989.

[7] A. Konar. *Artificial intelligence and soft computing: behavioral and cognitive modeling of the human brain.* CRC Press, Inc., Boca Raton, FL, USA, 2000.

[8] V. Kumar. Algorithms for constraint-satisfaction problems: a survey. *AI Mag.*, 13(1):32–44, 1992.

[9] H. T. Marin, R. de la Calleja-Manzanedo, and M. Valenzuela-Rendón. Genetic algorithms for dynamic variable ordering in constraint satisfaction problems. *Research in computing science.*

[10] D. E. Moriarty and R. Miikkulainen. Evolutionary neural networks for value ordering in constraint satisfaction problems. Technical Report AI94-218, The University of Texas at Austin, 1994.

[11] D. E. Moriarty and R. Miikkulainen. Hierarchical evolution of neural network. *IEEE Magazine*, pages 428–433, 1998.

[12] P. Prosser. Binary constraint satisfaction problems: Some are harder than others. pages 95–99, 1994.

[13] R. Rojas. *Neural Networks: A Systematic Introduction.* Springer, July 1996.

[14] S. J. Russell and P. Norving. *Artificial Intelligence A Modern Approach.* Pretincel Hall, 1995.

[15] B. M. Smith. Locating the phase transition in constraint satisfaction problems. In *Artificial Intelligence*, 1994.

[16] K. O. Stanley and R. Miikkulainen. Efficient evolution of neural network topologies. *Proceedings of the 2002 Congress on Evolutionary Computation (CEC '02)*, pages 1757–1762, 2002.

[17] W. M. Taylor, P. Cheeseman, P. Cheeseman, B. Kanefsky, and B. Kanefsky. Where the really hard problems are. In *In J. Mylopoulos and R. Reiter (Eds.), Proceedings of 12th International Joint Conference on AI (IJCAI-91),Volume 1*, pages 331–337. Morgan Kauffman, 1991.

[18] H. Terashima-Marín, J. C. Ortiz-Bayliss, P. Ross, and M. Valenzuela-Rendón. Hyper-heuristics for the dynamic variable ordering in constraint satisfaction problems. In *GECCO '08: Proceedings of the 10th annual conference on Genetic and evolutionary computation*, pages 571–578, New York, NY, USA, 2008. ACM.

[19] D. E. P. K. Tsang and D. C. J. Wang. A generic neural network approach for constraint satisfaction problems. pages 12–22, 1992.

[20] E. Tsang. *Foundations of Constraint Satisfaction.* Academic Press,, 1993.

[21] C. F. Zárate. Hiperheurísticas mediante un algortimo genético con cromosomas de longitud variable para resolver problemas de corte de material en dos dimensiones. *Masters thesis, Instituto Técnologico de Estudios Superiores de Monterrey*, 2005.