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

OPTIMIZADOR DE PARÁMETROS BASADO EN QPSO PARA AUTONOMOUS SEARCH

DIEGO ANDRÉS MORAGA LIPPIANS
ROBERTO IGNACIO MORALES AGUIRRE

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

Diciembre, 2012

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

OPTIMIZADOR DE PARÁMETROS BASADO EN QPSO PARA AUTONOMOUS SEARCH

DIEGO ANDRÉS MORAGA LIPPIANS
ROBERTO IGNACIO MORALES AGUIRRE

Profesor Guía: Ricardo Soto De Giorgis
Profesor Correferente: Broderick Crawford Labrín

Carrera: Ingeniería de Ejecución en Informática

Diciembre, 2012

Dedicatoria

A mi familia por estar siempre presente y ser la base de lo que soy ahora, dándome toda mi educación, tanto académica, como valórica, por su incondicional apoyo en todo momento sin esperar nada a cambio. A mis amigos que siempre me han entregado ánimos y buenos momentos para hacer ameno el trabajo.
Todo este trabajo ha sido posible gracias a ellos.
Diego Moraga Lippians.

A mis padres Chrisitan Morales y Mercedes Aguirre. por otorgarme sustento, una formación impecable y apoyo constante, con perseverancia, firmeza, y sobre todo, paciencia. A mis tíos Patricio Jorquera, Luis Zamora y Rosa Aguirre por ayudarme incondicionalmente en situaciones difíciles y creer en mis capacidades.
Roberto Morales Aguirre.

Agradecimientos

A mi familia y amigos sin los cuales no hubiese sido posible terminar este trabajo, a mi profesor guía por incentivar y motivar en todo el proceso del desarrollo del proyecto, a mi compañero Roberto Morales por ser un apoyo en todo el proceso de investigación, a Pablo Cuevas, Sebastián Sandoval, Miguel Blanco, Luis Alarcón, Cristóbal González por ayudarme en el proceso de correcciones.
Diego Moraga Lippians.

Índice

Lista de Figuras

Lista de Tablas

Resumen

Los Problemas de Satisfacción de Restricciones son aquellos problemas que se expresan por un conjunto de variables y restricciones. Una solución a un CSP estará dada por la instanciación de todas las variables, donde ninguna restricción sea violada. La metodología Autonomous Search (AS) se encarga, dentro de la programación con restricciones, de agilizar el proceso de resolución. Básicamente la idea consta en reemplazar aquellas estrategias de resolución que aparenten no mostrar un buen rendimiento por otras más prometedoras. El reemplazo de estrategias es controlado por una función de elección que actualmente es optimizada por medio de un algoritmo genético.

El objetivo de este proyecto es reemplazar el algoritmo genético por un algoritmo basado en QPSO (Quantum Particle Swarm Optimization) con el fin de determinar si este cambio resulta en una mejora en la efi ciencia de la arquitectura para AS.

Palabras clave *Autonomous Search, Constraint Satisfaction Problems, Enumeration Strategies, Choice Functions.*

Abstract

Constraint Satisfaction Problems (CSP) are those problems that are expressed by a set of variables and constraints. A solution to a CSP is given by the instantiation of all the variables, where no constraint is violated. Autonomous Search (AS), in Constraint Programming, is devoted to accelerate the resolution process of CSPs. Basically the idea is to replace those stratagies exhibiting a bad performance by more promising ones. The replacement of strategies is controlled by a choice function that currently is optimized by means of a genetic algorithm.

The purpose of this project is to replace the genetic algorithm by a QPSO (Quantum Particle Swarm Optimization) algorithm with the aim of determining if this change results in a better performance for the AS architecture.

Key-Words *Autonomous Search, Constraint Satisfaction Problems, Enumeration Strategies, Choice Functions.*

1. Introducción

La programación con restricciones, en inglés Constraint Programming (CP), es un paradigma de la programación en informática utilizado para la resolución de ecuaciones, donde las relaciones entre las variables son expresadas en forma de restricciones. Los problemas son formulados como "Problemas de satisfacción de restricciones", en inglés Constraint Satisfaction Problem (CSP), donde los problemas matemáticos son definidos como un conjunto de objetos que deben satisfacer un cierto número de restricciones o limitaciones. CP es extensamente usado en diferentes áreas de investigación, tales como inteligencia artificial, investigación de operaciones, bases de datos, sistemas expertos, etc.

La presente investigación se basa en una metodología llamada Autonomous Search (AS), el cual posee la característica de permitir a los sistemas mejorar su rendimiento mediante su propia adaptación o por adaptación supervisada. Este enfoque se orienta a la toma de decisiones en una fase clave para resolver CSPs, llamada fase de enumeración. En la fase de enumeración, existen dos importantes decisiones: El orden en el cual las variables son seleccionadas y el orden en que los valores son seleccionados. Esta selección se refiere al ordenamiento heurístico de variable y valor, que conjuntamente constituyen la estrategia de enumeración.

El objetivo de usar AS en CP es desarrollar una adaptación propia del proceso de búsqueda cuando éste exhibe un rendimiento bajo. Esto consiste en desarrollar el reemplazo "en el vuelo" de aquellas heurísticas que exhiben un bajo rendimiento por otras más prometedoras. El framework actual permite calcular la calidad de las estrategias a través de una Choice Function. La Choice Function comprueba el funcionamiento de la estrategia con la que se está iterando en un tiempo determinado, esto gracias a un conjunto de parámetros de control e indicadores que se van generando durante el proceso de ejecución.

Actualmente para determinar el mejor conjunto apropiado de parámetros para la choice function, se utiliza un algoritmo genético (GA) que evalúa y evoluciona diferentes combinaciones de parámetros para poder encontrar la solución a los problemas de CP, aliviando la tarea de parametrización manual.

El presente proyecto tiene como objetivo remplazar el algoritmo genético por el algoritmo QPSO, el cual busca determinar qué estrategia utilizar en cada paso de la ejecución de un problema, mediante procesos que imita el actuar de un enjambre de insectos, un banco de peces o aves en bandadas. El algoritmo QPSO determinará qué parámetros son los más idóneos para

determinar una respuesta a un problema.

La organización del informe es la siguiente: comenzando por la introducción y un breve resumen de lo que abarcará el proyecto, se indicarán los objetivos y el estado del arte. Los primeros capítulos presentarán los temas necesarios para poder comprender el tema de investigación, comenzando por que es la programación con restricciones, (resolución, formalización y ejemplos de problemas de satisfacción de restricciones), los algoritmos de búsqueda y los tipos de estrategias de resolución para llegar a las soluciones de estos problemas. En el capítulo siguiente se hará una breve explicación de los solvers, destacando el que se usará en la investigación (ECLiPSe). Luego en el siguiente capítulo se define Autonomous Search y su arquitectura, dando pie para describir la implementación de la choice function, y así adentrarse en el objetivo principal de la investigación, en el capítulo 11 se especificará con detalle el proceso de implementación del optimizador en la choice function. En los capítulos siguientes se presentarán los experimentos realizados con la herramienta AS utilizando el algoritmo QPSO como optimizador, y para dar fin se definirán las conclusiones luego de terminar la investigación.

2. Objetivos

Objetivo general

- Implementar e Integrar un optimizador de parámetros basado en el algoritmo QPSO en la arquitectura de Autonomous Search.

Objetivos específicos

- Analizar el funcionamiento de la arquitectura para Autonomous Search implementada anteriormente.
- Implementar un algortimo basado en QPSO para optimizar parámetros en la función de elección de la herramienta Autonomous Search.
- Realizar pruebas en base a la herramienta de Autonomous Search con el optimizador QPSO.

3. Estado del arte

La Programación con Restricciones [4] es una de las principales contribuciones de las ciencias de la computación para resolver problemas de alta complejidad. Este paradigma es usado para representar una amplia variedad de problemas, entre los cuales podemos encontrar, desde simples juegos de lógica como el sudoku, hasta problemas de gran envergadura y complejidad, como los presentes en áreas de inteligencia artificial e investigación de operaciones, entre otros.

Los primero trabajos relacionados con la programación de restricciones datan de los años 60 y 70 en el campo de la Inteligencia Artificial, por ejemplo el estudio para procesar algoritmos y generar sintéticamente imágenes visuales para integrar o alterar la información visual y espacial tomada del mundo real, procesamiento digital de imágenes por ejemplo para sensores remotos.

Existen también trabajos realizados en CP, centrados en la definición de estrategias para especificar clases de problemas, como por ejemplo, asignación de job shop [14], [16] o configuración del diseño [13], también se pueden encontrar trabajos realizados en CP centrados en establecer la mejor estrategia basada en algunos criterios estáticos [8], [17], [9]. Sin embargo, resulta bastante difícil tomar una decisión a priori, al igual que predecir su efecto.

Autonomous Search siendo una herramienta relativamente nueva ha pasado por muchos procesos de cambios [2], [1], los cuales sin modificar su idea original, han desarrollado una mayor eficiencia en su capacidad de procesamiento. Originalmente AS partió operando bajo un framework que recolectaba información durante la búsqueda para hacer decisiones correctas entorno a las estrategias de enumeración en el momento de la ejecución del problema. La toma de decisiones en el desarrollo de los procesos, correspondían a elementos claves de Autonomous Search, que corresponden a indicadores, choice functions y snapshots [1] [6], actualmente se ha completado el estudio para permitir un proceso modularizado [18] en la herramienta AS lo que permite un desarrollo mucho más eficiente a la hora de resolver un problema de satisfacción de restricciones.

[1] Los snapshots son observaciones de la resolución del CSP bajo AS. Son observaciones no continuas (fotografías) de los datos obtenidos.

4. Programación con restricciones

Los primeros trabajos de la programación con restricciones datan de los años 60's, en el campo de la resolución de problemas orientados a la inteligencia artificial, luego en los 70's se define la programación lógica (Prolog), en los años 80's ya con más estudios se define la programación lógica con restricciones (CLP).

La programación con restricciones resulta ser una paradigma de la programación en informática, donde las relaciones entre las variables son expresadas en términos de restricciones (ecuaciones), este paradigma es basado en la especificación de un conjunto de restricciones, las cuales deben ser satisfechas por cualquier solución del problema planteado, en lugar de especificar los pasos para obtener dicha solución. El enfoque de la programación con restricciones se basa principalmente en buscar un estado en el cual una gran cantidad de restricciones sean satisfechas simultáneamente. Un problema se define típicamente como un estado de la realidad en el cual existe un número de variables con valor desconocido. Un programa basado en restricciones busca dichos valores para todas las variables.

Para la resolución de un problema de satisfacción con restricciones se deben tener en cuenta las dos etapas, primero está la fase de modelado del problema, donde se definen variables, dominios y restricciones, una vez lista esta etapa se procesa con los algoritmos de búsqueda y métodos de consistencia, que lo implementará el Solver.

4.1. Estudiar las bases de la programación con restricciones

La idea es la de resolver problemas mediante la especificación de restricciones (condiciones y propiedades) que debe satisfacer la solución. Se desea obtener:

- Una solución, sin preferencia alguna.
- Todas las soluciones.
- Una óptima o al menos una que cumpla con los requerimientos, dando alguna función objetivo definida en términos de algunas o todas las variables.

4.2. Problema de satisfacción de restricciones

4.2.1. Resolución de un problema de satisfacción de restricciones

La resolución de un problema de satisfacción de restricciones (CSP) consta de dos fases diferentes:

modelar el problema como un problema de satisfacción de restricciones. La modelización expresa el problema por medio de una sintaxis de CSPs, es decir, mediante un conjunto de variables, dominios y restricciones del CSP.

Procesar el problema de satisfacción de restricciones resultante. Una vez formulado el problema como un CSP, hay dos maneras de procesar las restricciones: Las técnicas de consistencia que intentan eliminar valores inconsistentes en los dominios de las variables y algoritmos de búsqueda, que se basan en la exploración sistemática del espacio de soluciones hasta encontrar una solución o probar que no existe tal solución.

4.2.2. Formalización de un problema de satisfacción de restricciones

Un problema de satisfacción de restricciones se representa como una terna (X, D, C) donde: X es un conjunto de variables de tamaño n, tal que $x_1, x_2, ...x_n$. D es una tupla de dominios finitos $d_1, d_2, ..., d_n$. Es el dominio que contiene todos los posibles valores que se le pueden asignar a la variable x_i.

C es un conjunto finito de restricciones. Cada restricción m-aria (c_m) está detenida sobre un conjunto de variables $(x_1, x_2, ..., x_n)$restringiendo los valores que las variables pueden simultaneamente tomar.

Una solución a un CSP es una tupla consistente que contiene todas las variables del problema. Una solución parcial es una tupla consistente que contiene algunas de las variables del problema. Un CSP es consistente, si tiene al menos una solución, es decir una tupla consistente. Dos CSPs son equivalentes si ambos representan el mismo conjunto de soluciones.

4.2.3. Ejemplo de CSP (N-reinas)

Consiste en posicionar en un tablero de ajedrez de tamaño NxN, una cantidad N de reinas evitando que, con sus movimientos característicos en el juego de ajedrez, se amenacen entre ellas, esto quiere decir que, dada la posición en la que se encuentren, no les sería posible atacar a otra reina. Cuando N es un número pequeño el costo de realizar una búsqueda de la solución con un algoritmo de búsqueda de baja complejidad como generate and

test es menor ya que con un tablero de 4x4 existen 256 distintas configuraciones, pero con n = 16 existen 18,446,744,073,709,551,616 configuraciones. Una solución para el problema teniendo N = 4 sería:

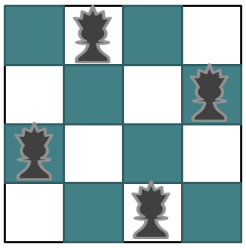

Figura 1: Problema n reinas, con N = 4.

Este problema puede ser formulado como:

- Variables: $\{x_i\}, i = 1...N$.
- Dominio $= \{1, 2, 3, ..., N\}$, para todas las variables.
- Restricciones: $(\forall x_i, x_j, i \neq j)$:
  - $x_i \neq x_j$, No en la misma fila.
  - $x_i - x_j \neq i - j$, No en la misma diagonal SE.
  - $x_j - x_i \neq i - j$, No en la misma diagonal SO.

4.2.4. Ejemplo de CSP (Send + More = Money)

La criptografía es una técnica utilizada para cifrar cadenas de caracteres y hacerlas inteligibles, un problema conocido es Send + More = Money, el cual consiste en descifrar el resultado numérico de esta ecuación según las reglas establecidas: cada letra tiene un equivalente numérico entre 0 y 9. Una letra puede representar sólo un dígito. Dadas estas reglas pueden definirse las restricciones necesarias para poder resolver este problema.

Figura 2: Problema Send+More=Money.

La manera más fácil de modelar este problema es asignando una variable a cada una de las letras, todas ellas con un dominio $\{0, ..., 9\}$ y con las restricciones de que todas las variables toman valores distintos y con la correspondiente restricción para que se satisfaga 'send+more=money'. De esta forma las restricciones (no binarias) son:

- $10^3(s+m)+10^2(e+o)+10(n+r)+d+e = 10^4m+10^3o+10^2n+10+y$;
- Todas las letras deben representarar dígitos distintos (s, e, n, d, m, o, r, y);

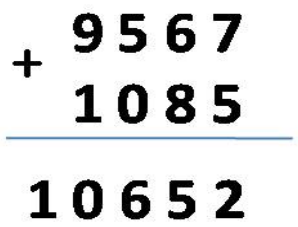

Figura 3: Problema Send+More=Money resuelto.

5. Algoritmos de búsqueda

Son aquellos que están diseñados para explorar el espacio de estados de problema exhaustivamente hasta encontrar una posible solución que cumpla de forma simultánea con todas las restricciones del problema. Estos algoritmos están garantizados para encontrar una solución, si existe, o para demostrar que el problema es insoluble. La desventaja de estos algoritmos es que pueden tardar mucho tiempo en hacerlo. Una definición más clara del concepto del algoritmo de búsqueda la da el siguiente esquema:

- Explora el espacio de estados del problema (posibles configuraciones).
- Condiciones de término del algoritmo:
  - Encontrar una solución.
  - Demuestra que no existe una solución.
  - Agota los recursos computacionales.

Es llamado un árbol de búsqueda (figura 4) a un espacio generado por las posibles combinaciones de la asignación de valores en un CSP representado por niveles, este está formado por un nodo raíz llamado padre también conocido como tupla vacía ya que en su interior no se encuentra valor alguno, suponiendo que se trabaja con tres variables se escribiría de la siguiente manera (-,-,-). Los siguientes niveles son representados de esta manera: Primer nivel (1-tupla) = (0,-.-), segundo nivel (2-tupla) = (0,0,-), hasta definir el n-ésimo nivel (n-tuplas) definiendo n como el número de variables que representan el problema. Si una n-tupla es consistente, entonces es solución del problema. Un nodo del árbol de búsqueda es consistente si la asignación parcial actual es consistente, o en otras palabras, si la tupla correspondiente a ese nodo es consistente. A continuación se mostrarán los métodos de búsquedas más utilizados.

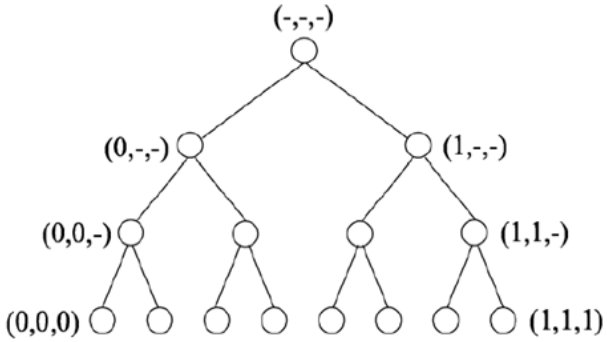

Figura 4: Ejemplo de ábol de Búsqueda.

5.1. Generate and test (GT)

Entre todos los métodos de búsqueda este es el más simple de todos, consiste en instanciar cada una de las variables, comprobando por cada una, hasta encontrar la solución del problema la cual sería la primera variable que cumpla con las restricciones originales. En este paradigma, cada combinación posible de las asignaciones de variables de manera sistemática se genera y se prueba para ver si cumple con todas las restricciones. GT es un algoritmo poco eficiente ya que genera una pérdida de rendimiento pues este efectúa comprobaciones innecesarias y repetidas. La desventaja de este algoritmo es que realiza muchas instanciaciones erróneas de valores a variables que después son rechazadas en la fase de verificación, ya que las restricciones se evalúan sólo si se encuentran todas las variables instanciadas. La solución para este algoritmo es ir evaluando las restricciones apenas se instancien las variables involucradas. Un ejemplo de Generate and Test es el que muestra la figura 5, el cual muestra el problema de las N-reinas según se ve sólo se evalúan las restricciones cuando se genera un verificación en cada una de las variables participantes. Es decir recorrer todos los niveles del árbol, por lo que en la mayoría de los casos se genera un alto costo utilizar este método que sólo es recomendable si se va a aplicar a problemas pequeños.

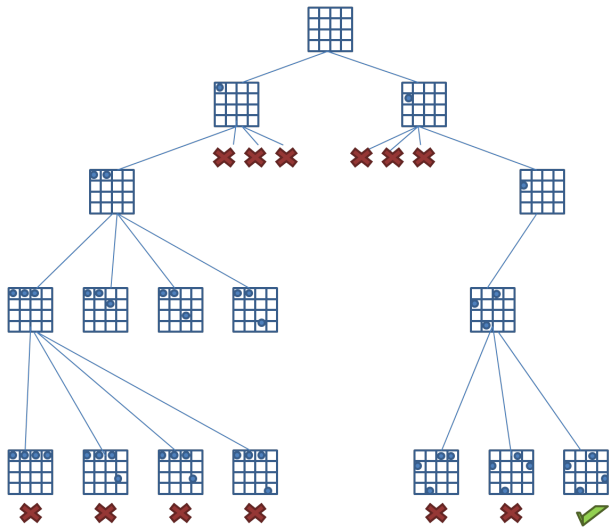

Figura 5: Ejemplo de Generate and Test utilizando el caso de las 4-reinas.

5.2. Backtracking cronológico (BT)

Este método (BT) es una mejora al algoritmo GT, trabaja realizando una exploración en profundidad del espacio de búsqueda, cada vez que se asigna un nuevo valor a la variable actual (x_i), se comprueba si es consistente con los valores que han sido asignados a las variables pasadas. Si es así, sigue con la instanciación de una nueva variable. En caso de conflicto, intenta asignar un nuevo valor a la última variable instanciada, si es posible, y en caso contrario retrocede a la variable asignada inmediatamente anterior y así sucesivamente hasta encontrar una asignación de un valor a una variable que es consistente con las variables pasadas o hasta que se demuestre que no existe solución que satisfaga el problema. Es decir, BT recorre el árbol utilizando búsqueda primero en profundidad y, en cada nodo, comprueba si la variable actual es consistente con las variables pasadas. Si se detecta inconsistencia se descarta la asignación parcial actual ya que no es parte de ninguna asignación completa que sea solución. De esta forma, se ahorra recorrer el subárbol que cuelga de esta asignación parcial. El backtracking cronológico es un algoritmo muy simple pero también muy ineficiente. Los problemas que este presenta son los siguientes: Sólo presenta una visión local del problema, de forma que sólo comprueba las restricciones entre las variables ya instanciadas, e ignora las relaciones entre dichas variables y las que quedan por instanciar. Este algoritmo no recuerda movimientos previos, por

lo que no asegura que no los pueda repetir. No se pueden detectar inconsistencias sin instanciar todas las variables involucradas en una restricción. En la figura 6, se muestra el proceso de búsqueda utilizando como ejemplo el problema de las 4-reinas, mediante BT, las fallas se detectan con sólo dos variables instanciadas.

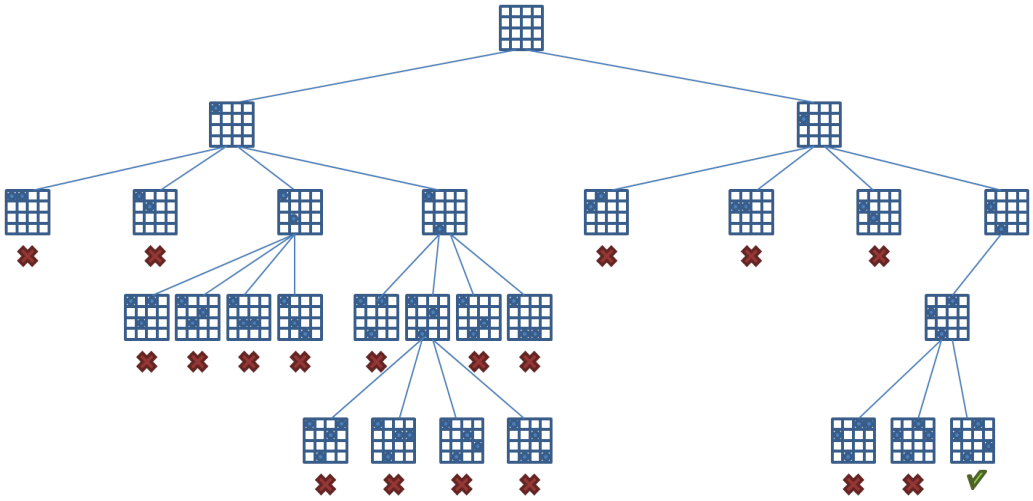

Figura 6: Ejemplo de Backtracking cronológico utilizando el caso de las 4-reinas.

5.3. Forward checking

Este algoritmo (FC), es de los llamados algoritmos Look-Ahead más comunes, que son algoritmos que hacen una comprobación hacia adelante en cada etapa de la búsqueda, es decir, en cada etapa de la búsqueda BT, FC comprueba hacia adelante la asignación actual con todos los valores de las futuras variables que están restringidas con la variable actual, mediante el uso de técnicas de consistencia (capítulo 7). Los valores de las futuras variables que son inconsistentes con la asignación actual son temporalmente eliminados de sus dominios. FC garantiza que en cada etapa, la solución parcial actual es consistente con cada valor de cada variable futura. Además, cuando se asigna un valor a una variable, solamente se comprueba hacia adelante con las futuras variables involucradas. Así mediante la comprobación hacia adelante, FC puede identificar antes las situaciones sin salida y podar el espacio de búsqueda. Si el dominio de una variable futura se quedó vacío, la instanciación de la variable actual se deshace y se prueba con un nuevo

valor. Por lo tanto, el algoritmo, al detectar que la actual solución es inconsistente, elimina las búsquedas del sub-árbol usando un simple Backtracking Cronológico.

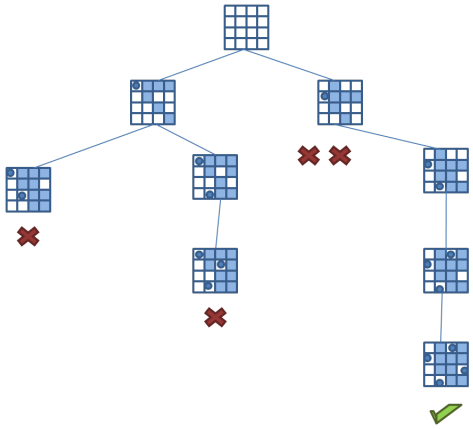

Figura 7: Resolución problema 4-reinas con Forward checking.

5.4. Maintaining arc consistency

El procedimiento realizado en el algoritmo FC intenta reducir el espacio de búsqueda de forma temprana a través de un proceso más inteligente, capaz de instanciar los conflictos actuales y los futuros. No obstante, hace un trabajo mayor en la asignación de cada variable. El proceso Maintaining Arc Consistency es descrito en la figura 8. Este metodo trabaja eliminando los espacios de conflictos actuales y futuros ulitizando técnicas de consistencia arco explicadas en la capítulo 7. Como se muestra en la imagen, al encontrar a la reina en la posición (1,1) los conflictos entre aquella celda y las futuras son eliminadas, mediante la acotación del dominio de las posiciones posteriores, (indicado con las líneas azules). Luego de eso, desde la posición (1,1), se comienza buscando la primera posición disponible en la segunda columna, el cual corresponde a la celda (3,2). El proceso algorítmico, descubre que aquella posición es inconsistente, ya que desde esa ubicación, no existe lugar para una tercera reina (indicado con las líneas naranjas), de esta forma, la ubicación (3,2) es removida. A continuación el algoritmo, sigue con la celda (4,2), que corresponde a la segunda posición disponible, de la segunda columna. Este posicionamiento establece que la única celda disponible para la tercera reina, debe ser la (2,3), la cual eventualmente provocaría una inconsistencia, ya que no habría lugar disponible para una cuarta reina. El

ejemplo del funcionamiento del algoritmo para la primera posición, se repite de forma equivalente, hasta llegar al resultado de la derecha del árbol de la figura 8.

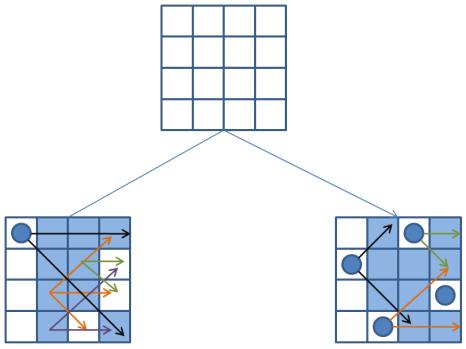

Figura 8: Resolución problema 4-Reinas utilizando Maintaining Arc Consistency.

6. Estrategias de resolución

6.1. Técnicas de Consistencia

Las técnicas de consistencia, son por lo general algoritmos utilizados con la finalidad de disminuir el espacio de búsqueda mediante un proceso de filtrado y así poder simplificar el problema, el proceso de filtrado elimina elementos que, con seguridad no pueden ser parte de la solución. Dichas técnicas, se clasifican en distintos niveles.

6.1.1. Consistencias de nodo

La consistencia local más simple de todas es la consistencia de nodo o nodo-consistencia, esta se basa en asegurar que todos los valores en el dominio de una variable satisfagan las restricciones unarias sobre esa variable. Por lo tanto, se dirá que un CSP es nodo-consistente si y sólo si cada una de sus variables es nodo-consistente, todos aquellos valores que violen alguna restricción unaria se eliminan del dominio ya que estos valores no se encontrarían contenidos en ninguna solución. De manera que:

$$\forall x_i \in X, \forall c_i \in C, \exists a \in D_i : (\alpha) \in c_i$$

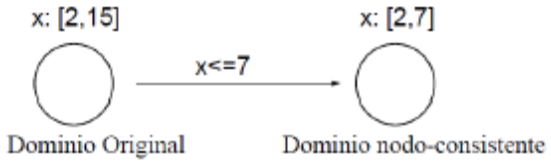

Figura 9: Consistencia de nodos.

6.1.2. Consistencias de arco

La consistencia de arco trabaja con restricciones binarias. El arco es consistente si: para cualquier valor del dominio asociado a la variable distinguida del arco, existen valores en los dominios de las restantes variables que satisfacen la restricción del arco, es decir, un problema binario es arco-consistente si para cualquier par de variables restringidas X_i y X_j, para cada valor α en D_i hay al menos un valor β en D_j tal que las asignaciones (X_i, α) y (X_j, β) satisfacen la restricción entre X_i y X_j. Cualquier valor en el dominio D_i de

la variable X_i que no es arco-consistente puede ser eliminado de D_i ya que no puede formar parte de ninguna solución [4]. El dominio de una variable es arco-consistente si todos sus valores son arco-consistentes.

6.2. Heurísticas y estrategias de enumeración

Los algoritmos de búsqueda requieren un orden en que se van estudiar las variables, así como el orden en que se van a instanciar los valores de cada una de las variables implicadas. El hecho de escoger correctamente como abordar las variables a estudiar reducirá considerablemente el espacio de búsqueda.

Las Estrategias de Enumeración (Heurísticas de Selección de Variables y Valores) pueden ser usadas para reducir el costo de búsqueda en el proceso de resolución.

Encontrar una solución a un CSP que dé satisfacción plena a todas las restricciones involucra necesariamente llevar adelante un proceso de búsqueda bastante costoso. En este proceso de búsqueda, específicamente en la fase de enumeración, la estrategia utilizada como guía tiene un efecto importante en el rendimiento del proceso de resolución. Para visualizar mejor la importancia de las estrategias de enumeración, en la Figura 10 se muestran los árboles de búsqueda de la resolución del problema 10-Reinas utilizando tres distintas estrategias de enumeración.

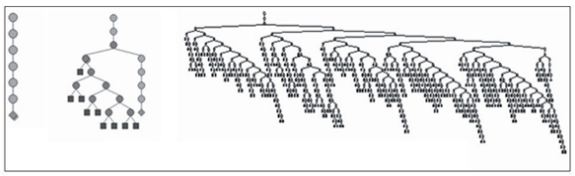

Figura 10: 10 Reinas Resuelto con 3 Estrategias Diferentes.

En la Figura anterior es posible ver que la primera estrategia utilizada encuentra inmediatamente la solución. La segunda estrategia, después de algunas malas elecciones (de variable y valor), finalmente obtiene directamente una solución; en cambio, la tercera estrategia de enumeración realiza numerosas malas elecciones antes de obtener una solución.

Observando esto, es obvio que diferentes estrategias tienen rendimientos significativamente diferentes, por lo cual es crucial seleccionar una buena estrategia de enumeración. Sin embargo, ésta no se puede predecir de manera general, y es por esto que se considera relevante observar el proceso

de resolución, identificar y clasificar distintos estados de dicho proceso en base a indicadores significativos, para con esta información poder reaccionar oportunamente frente a una mala decisión.

6.2.1. Heurísticas de ordenación de variables estáticas

Producen un orden fijo de las variables antes de iniciar la búsqueda y las variables son seleccionadas en orden predefinido para su instanciación.

- Mínimum with: Impone en primer lugar un orden total sobre las variables, de forma que el orden tiene la mínima anchura, y entonces selecciona las variables en base a ese orden. La anchura de la variable x es el número de variables que están antes de x, de acuerdo a un orden dado, y que son adyacentes a x. La anchura de un orden es la máxima anchura de todas las variables bajo ese orden. De esta forma las variables que están al principio de la ordenación son las más restringidas, y dejan al final las variables menos restringidas. Asignando las variables más restringidas al principio, las situaciones sin salida se pueden identificar antes y además se reduce el número de vueltas atrás.

- Máximum degreed: Esta heurística selecciona las variables decrecientemente según su grado con el grafo de restricciones original. Se entiende como grado de una variable al número de variables con las que está conectada. De esta forma, esta heurística selecciona primero las variables de mayor grado, es decir, las más conectadas. Min domain: Esta heurística selecciona las variables de acuerdo a la menor cardinalidad de su dominio. Las variables con dominio más pequeño son elegidas antes. Sin embargo, respecto a la aplicación de esta heurística, es preferible su versión dinámica (minimunremainingvalues).

6.2.2. Heurísticas de ordenación de variables dinámicas

Los algoritmos de ordenación de variables estáticos no consideran los cambios en los dominios o relaciones de las variables causados por la propagación de restricciones durante la búsqueda. Debido a esto se encuentran generalmente en algoritmos de comprobación hacia atrás, donde no se lleva a cabo la propagación de restricciones.

La heurística de ordenación de variables dinámicas más común se basa en el principio de primer fallo (FF) que sugiere que para tener éxito deberíamos intentar primero donde sea más probable que falle. De esta manera las situaciones sin salida pueden identificarse antes y además se ahorra espacio

de búsqueda. De acuerdo con el principio de FF, en cada paso, seleccionaríamos la variable más restringida. La heurística FF también conocida como heurística Minimum Remaining Values.

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

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

- Maximum Forward Degree: Selecciona una variable con el número más largo de vecinos (variables cuyos nodos son adyacentes en el grafo de restricciones) en el conjunto de variables no instanciadas.

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

6.2.3. Heurítica de selección de valor

Estas heurísticas seleccionan el valor con más probabilidades en el dominio para conducir al problema a una posible solución. Generalmente las heurísticas en su mayoría eligen un valor que este restringiendo menos a las actuales variables, con ello se deja una mayor cantidad de valores útiles para las siguientes variables. Min-Conflicts. Este tipo de heurística, selecciona el valor que provoca los mínimos conflictos para futuras asignaciones. Es decir, se cuenta la cantidad de valores del dominio de cada variable vecina, que son incompatibles con el valor seleccionado, posteriormente se elige a aquel cuya suma sea la más baja. En caso de que existiera más de un mínimo, se escogerá uno entre ellos de manera aleatoria.

- Survival. Este tipo de heurística es una variación respecto a la anterior, en este caso, el número de valores incompatibles dentro del dominio y luego es dividido por el tamaño de este. Esto da como resultado el porcentaje de valores útiles que pierde el dominio debido al valor

que se está examinando. Los porcentajes son sumados para todas las variables futuras con las que está relacionada la variable que se quiere instanciar. Se elige el valor con la suma más baja.

- Max Domain Size: Se elige el valor que deja los máximos tamaños de dominio para las futuras variables. En caso de que existan empates, existe otra heurística, denominada Weighted Max DomainSize, la cual específica la forma de elección en estos casos.

7. Solvers

Los Solvers tuvieron su primera incursión en los años 60, fueron creados para poder desarrollar y resolver CSPs, siendo usados hasta hoy en día para poder trabajar con problemas que involucran variables, dominios y restricciones. Para soportar restricciones y diversos paradigmas, han sido creados variados tipos de Solvers de diversa naturaleza.

7.1. Programación lógica con restricciones

La programación lógica con restricciones (CLP, Constraint Logic Programming) es el paradigma que une la programación lógica y la satisfacción de restricciones de un problema dado, por lo general estos lenguajes de programación con restricciones son ampliaciones de otro lenguaje y se caracterizan por poseer una estructura algebraica. Las funciones especiales y predicados simbólicos se interpretan sobre un dominio fijo, toda relación que se establezca con el dominio es denominada restricción. Esta idea fue impulsada por Colmerauer en el desarrollo de Prolog II, al ser prolog el primer lenguaje usado dio como efecto que este campo fuese llamado inicialmente Programación Lógica con Restricciones. Ambos paradigmas comparten características muy similares, tales como las variables lógicas (una vez que una variable es asignada a un valor, no puede ser cambiado).

La programación con restricciones puede ser implementada como un lenguaje propio o como bibliotecas para ser usadas en algún lenguaje de programación imperativo. Algunos ejemplos de Solvers que soportan CLP se nombran en las secciones siguientes.

7.2. ECLiPSe

ECLiPSe es el más reciente sistema CLP, provee librerías con la finalidad de tratar cierto tipo de problemas como los que involucran dominios continuos o programación matemática [12]. Tiene una gran variedad de características que permiten solucionar problemas con restricciones, como manejo de listas, arreglos y registros, además de un conjunto de instrucciones de control.

7.3. GNU prolog

Es otro sistema CLP [7], fue creado para trabajar problemas con dominios finitos, también se pueden usar para trabajar con números reales. Entre sus herramientas se encuentra una larga lista de predicados predefinidos de

Prolog y restricciones para el manejo de listas y estructuras condicionales. Se ha incluido soporte para problemas de optimización y heurísticas de ordenamiento, además de una interfaz para llamar a rutinas externas escritas en el lenguaje C.

7.4. Librerías

Las librerías generan un lenguaje necesario para enunciar problemas bajo restricciones al haber limitaciones con algún lenguaje de programación. Por lo general su implementación es llevada a cabo por medio de clases y métodos específicos. Por ejemplo, una determinada clase puede ser usada para definir el estado de las variables y los métodos, junto con las relaciones entre ellos. Esta es una manera de implementar un sistema de restricciones sin la necesidad de desarrollar un nuevo lenguaje de programación. De igual manera, el usuario está obligado a conocer el lenguaje original para la determinada librería que desee utilizar. Algunos Solvers que pertenecen a esta categoría se exponen a continuación.

7.4.1. Generic constraint development Eenviroment (Gecode)

Gecode [15] es una librería abierta escrita sobre el lenguaje C++, eficiente y portable para el desarrollo de sistemas y aplicaciones basados en restricciones. Fue diseñada para soportar variables de dominio finito. El conjunto de restricciones que abarca la librería, permite trabajar sobre tipos de datos enteros, booleanos y conjuntos de variables, soporta la especificación de heurísticas de ordenamiento de variable y de selección de valor. Entre los lenguajes en los cuales se han desarrollado interfaces para trabajar con librerías Gecode se encuentran Prolog (YAP Prolog), Java (Gecode/J), Ruby (Gecode/R), Lisp (GeLisp) y Phyton.

7.4.2. Koalog

Koalog CLP [11] es una librería que trabaja en base al lenguaje Java la cual permite trabajar con problemas de satisfacción y optimización de restricciones. Admite definir conjuntos de restricciones y dominios, ambos finitos. Entre sus cualidades, soporta la definición de heurísticas de selección de variable y de valor.

7.4.3. Choco

Es una librería desarrollada en Java. Provee un variado conjunto de restricciones para ser aplicadas sobre números enteros, reales, y conjunto de variables. Es capaz de soportar problemas de optimización, el proceso de búsqueda puede ser utilizado por estrategias de selección de variables y valor, predefinidas por el Solver o definidas por el usuario.

8. Aspectos del lenguaje ECLiPSe [23]

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

8.1. Lenguaje de programación ECLiPSe

ECLiPSe es un software de código abierto, de desarrollo de sistemas y despliegue de aplicaciones de programación con restricciones.

Este lenguaje posee múltiples bibliotecas para solucionar problemas de programación con restricciones, un modelamiento de alto nivel y un control del lenguaje. Es una plataforma lógica de programación altamente declarativa. Una de las razones para escoger el lenguaje de ECLiPSe es la particularidad de poseer múltiples características que facilitan el desarrollo de un Solver CSP, características y especificaciones que hacen a este lenguaje uno de los más apropiados para este tipo de programas. Otras razones son, que existe abundante información relacionada al uso de ECLiPSe, permite la gestión de archivos, permite el uso de ciclos de repetición, usados en los lenguajes de programación tradicionales. El enfoque de Programación Lógica con Restricciones (CLP), permite la unión natural de dos paradigmas declarativos, Programación lógica[2] y Programación con Restricciones[3], esta combinación ayuda a hacer programas expresivos y flexibles, y en algunos casos más eficientes que otros programas, dado que reducen dramáticamente el tiempo de ejecución mientras logran una eficiencia similar a los lenguajes procedimentales.

Las principales características de ECLiPSe consisten en, un núcleo de tiempo de ejecución, una colección de bibliotecas, un modelado y control de lenguaje, un entorno de desarrollo, interfaces para integrarse en un entorno host e interfaces para resolucionadores anexos, además una característica importante en ECLiPSe, es que provee distintas formas de crear ciclos, evitando repeticiones por escrito en forma de predicados recursivos y que permite el manejo de archivos tanto de entrada como de salida, lo cual facilita enormemente las distintas pruebas.

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

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

Algunas funciones con las que trabaja ECLiPSe son las siguientes:

- **Foreach(X, Lista)**: Iterar objetivos con X abarca todos los elementos que contiene la Lista. También se puede usar para la construcción de una lista. X es una variable local en las metas.
- **For(I, MinExpr, MaxExpr)**: Itera objetivos con I, recorre un rango de enteros que abarcan desde MinExpr a MaxExpr estos rangos pueden ser expresiones aritméticas. Se restringe su uso solamente para control de iteraciones, el incremento predeterminado de la iteración es 1. I es una variable local en objetivos.
- **For(I, MinExpr, MaxExpr, Increment)**: Tal como se definió el anterior pero el incremento es especificado.
- **Fromto(First, In, Out, Last)**: La iteración es comenzada con In = First y termina hasta que Out = Last. In y Out son variables locales.
- **Count (I, Min, Max)**: Itera sobre enteros desde Min hasta Max. I es una variable local. Puede ser usada para controlar iteraciones así como también para contar.

Otra de las características importantes de ECLiPSe, es que permite el manejo de archivos tanto de entrada como de salida, lo cual facilita en gran medida el trabajo requerido al realizar las distintas pruebas.

9. Autonomous search

Autonomous Search puede ser definido como una herramienta que se encarga de agilizar el proceso de búsqueda de una solución a problemas de programación con restricciones (CSP). Para realizar esto la herramienta realiza una serie de procesos para determinar de qué manera o qué decisiones tomar para llegar a una posible solución rápidamente. Básicamente las distintas decisiones son evaluadas durante el proceso de búsqueda y los valores de estas evaluaciones son almacenados, luego, se comparan estas evaluaciones para determinar cuáles son las estrategias se adaptan mejor a la naturaleza del problema que se está resolviendo

Autonomous Search es un caso particular de sistemas adaptativos cuyo objetivo es mejorar el proceso de resolución adaptándose a sí mismo al problema en cuestión [19]. Se caracteriza por modificar componentes internos cuando es expuesto a fuerzas externas variables. Los componentes internos son algoritmos involucrados en el proceso de búsqueda-heurísticas, inferencias, etc. Las fuerzas externas corresponden a la evolución de la información recolectada en el proceso de búsqueda. Esta información puede ser tanto directamente recogida en el problema como computarizada indirectamente a través de la eficiencia percibida de componentes individuales. La información recogida incluye, por ejemplo, el tamaño del espacio de búsqueda (exacto o estimado), el número de sub-problemas, etc. La información computarizada incluye el grado de discriminación de heurísticas, la capacidad de podamiento en técnicas de inferencia, etc.

9.1. Componentes de Autonomous search

Con tan sólo seleccionar una heurística de selección de variable o valor para encontrar solución a un problema de satisfacción de restricciones, no se asegura el hecho de encontrar una solución a este, y de ser así, no afirma que esta sea óptima, o que el resultado se obtenga eficientemente, ante esto, es necesario un proceso de resolución para evaluar las estrategias a utilizar y que representarán el camino para encontrar la solución o parte de este.

Encontrar la alternativa a elegir requiere de un trabajo previo para iniciar el proceso de resolución denominado sampling, el cual se encarga de probar cada estrategia de enumeración para almacenar estadísticas de su comportamiento en base a indicadores, los cuales son usados para determinar prioridades de estrategias a seguir y de esta forma establecer con cual es recomendable iniciar para resolver el problema.

La figura 11 representa el proceso de resolución:

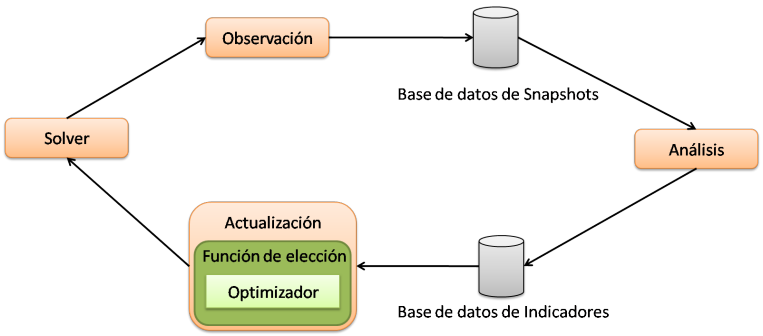

Figura 11: Esquema del framework actual [3].

Los procesos Solver, Observación y Análisis son procesos que influyen directamente en la resolución:

El componente Solver inicia con un algoritmo de resolución genérico que realiza una búsqueda profunda alternando programación de restricciones con fases de enumeración. Posee un conjunto de estrategias de enumeración básicas, las cuales irán actualizándose mientras se realiza el cálculo.

En el proceso de Observación se obtiene información referente al árbol de búsqueda con el que se está iterando observada mientras el componente Solver trabaja. Las observaciones, denominadas Snapshots, se van realizando periódicamente y se almacenan para ser revisadas como una abstracción del estado del proceso de resolución en un momento dado.

El componente de Análisis, en base a las capturas del proceso de observación explicado anteriormente, analiza las estrategias y a partir de ellas genera indicadores que utilizará el componente de actualización. Estos indicadores serán utilizados como referencia para evaluar el desempeño de la heurística estudiada.

El componente Actualización, a partir de los indicadores almacenados en la base de datos de indicadores que puede ser apreciada en la tabla 1, intenta establecer en un determinado tiempo la estrategia que conviene usar, mediante la utilización de un algoritmo genético, el cual calcula los parámetros de control.

Indicadores Fitness	Descripción
Step	Cada vez que una variable es instanciada por enumeración
N	Número de nodos visitados
VU	Número de variables que no son instanciada
SB	Shallow Bracktrack: cuando se trata de instanciar un valor a una variable sin éxito, entonces este recorre al próximo valor
B	Bracktracks, Si la actual variable lleva a una ruta sin salida, entonces el algoritmo lleva a la variable previa
VF	Número de variables instanciadas por enumeración y propagación
VFES	Número de variables instanciada por numeración en cada paso
TAVFES	Número total acumulado de variables instanciadas por enumeración en cada paso
PTAVFES	Porcentaje del total acumulado de variables instanciadas por enumeración en cada paso sobre el total de variables
VFPS	Número de variables instanciadas por propagación en cada paso
TAVFPS	Número total acumulado de variables instanciada por propagación en cada paso
PVFPS	Porcentajes de variables instanciadas por propagación en cada paso
PTAVFPS	Porcentaje del total acumulado de variables instanciadas por propagación en cada paso sobre el total de variables
TSB	Número total de Shallow Backtrack
D	Profundidad actual del árbol de búsqueda
DMAX	Profundidad máxima en el árbol de búsqueda
LN3	Reducción del espacio de búsqueda. Si es positivo, el espacio de búsqueda es menor que el de la captura
DB	Si la actual variable lleva a una ruta sin salida, entonces el algoritmo lleva a la variable previa, y cuenta el retroceso que existió
LN1	Profundidad máxima actual menos la profundidad máxima previa. Variación de la profundidad previa y actual
LN2	Es la profundidad actual menos profundidad previa. Cuando resulta positivo significa que el nodo actual está más profundo que el anterior
PVFES	Porcentaje de Variables instanciadas por enumeración en cada paso

Tabla 1: Ejemplo de tabla de indicadores [10]

10. Choice function

La tarea de la función de elección (Choice Function) es evaluar el rendimiento de las estrategias de enumeración en base a los valores de los indicadores observados para cada estrategia en un instante de tiempo determinado. Como su nombre lo indica, La función de elección se encarga de escoger cuáles son las estrategias que muestran un mejor resultado utilizando los valores entregados por los indicadores como referencia.

Para cualquier estrategia de enumeración S_j, la Choice Function f en el paso n por cada estrategia S_j está definida por la ecuación (1), donde l es el número de indicadores considerados y α es un parámetro para controlar la relevancia del indicador dentro de la Choice Function.

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

Para poder representar correctamente el peso o relevancia de un indicador i en una estrategia S_j se utiliza una técnica estadística llamada suavizado exponencial, esta técnica es usada para determinar la depreciación de los pesos de observaciones capturadas en instantes previos. El factor de suavizado está dado por β y su dominio es entre 0 y 1, β es aplicado al cálculo de $f_{i_n}(S_j)$ que se expresa en ecuaciones (2) y (3) y la variable x_0 indica el valor del indicador i de la estrategia S_j en el instante 1.

$$f_{i_1}(S_j) = x_0 \tag{2}$$

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

10.1. Optimizadores de parámetros

La Tarea de optimizador de parámetros dentro de la Choice Function, en términos de ecuaciones, es calcular el factor de relevancia representado por α que se encuentra en la ecuación (1). La variable α corresponde a la importancia que cada indicador muestra para cada una de las estrategias de enumeración evaluadas en un instante determinado.

Para determinar el conjunto más apropiado de parámetros para la choice function se usa un enfoque multinivel. Los parámetros son puestos a punto por el algoritmo QPSO (quantum-behaved particle swarm optimization)

que capacita los choice function llevando a cabo una fase de sampling. El sampling ocurre durante una fase de recolección de información inicial, donde la búsqueda es ejecutada repetidamente para fijar el cutoff (cierre), por ejemplo, bajo un número fijo de variables instanciadas, nodos visitados o backtracks. Después del muestreo, el problema es resuelto con el conjunto más apropiado de valores para el choice function. Hay que considerar que cuando las estrategias tienen el mismo resultado, se seleccionada una randómicamente. El algoritmo QPSO evalúa y actualiza diferentes combinaciones de parámetros, aliviando la tarea de parametrización manual. Cada miembro de la población permite obtener los parámetros de una choice function. Pero no por si solos, sino que cuando se ejecuta el QPSO. Luego, estos individuos son usados para crear una instancia choice function. Cada choice function instanciada (cada partícula) es evaluado en una fase de muestreo intentando resolver parcialmente el problema fácilmente para fijar el corte fijo (cutoff). Como un valor de evaluación para partícula, es usado un indicador del proceso de rendimiento (número de backtracks). Luego cada partícula de la población entra en evaluación, el enfoque multinivel es usado para ajustar la choice function, la choice function resultante es aplicada a resolver el problema CSP.

10.1.1. Particle swarm optimization

Los algoritmos de optimización de enjambre de partículas (PSO) fueron introducidos en 1995 por Kennedy y Eberhart como una alternativa a los algoritmos genéticos estándar. Estos algoritmos y está inspirada en el comportamiento social observado en grupos de individuos tales como parvadas de pájaros, los bancos de peces o los enjambres de insectos. Tal comportamiento social se basa en la transmisión del suceso de cada individuo a los demás individuos del grupo, lo cual resulta en un proceso sinergético que permite a los individuos satisfacer de la mejor manera posible sus necesidades más inmediatas. De hecho, los algoritmos basados en los principios del PSO buscan soluciones a un problema de optimización, al igual que el grupo de los animales que se encuentran en busca de comida o para evitar a depredadores. La metaheurística PSO ha demostrado ser muy eficiente a la hora de resolver problemas de optimización de un sólo objetivo con rápidas tasas de convergencia. Básicamente, la metaheurística PSO es un algoritmo iterativo el cual está basado en una población de individuos llamada enjambre, en la que cada individuo, denominado partícula, se dice que sobrevuela el espacio de decisión en busca de soluciones óptimas [5]. Así, dado un espacio de decisión N-dimensional, cada partícula i del enjambre conoce su posición actual

$X_i = [x_{i1}, x_{i2}, \ldots, x_{iN}]$, la velocidad $V_i = [v_{i1}, v_{i2}, \ldots, v_{iN}]$ con la cual ha llegado a dicha posición y la mejor posición $P_i = [p : i1, p_{i2}, \ldots, p_{iN}]$ en la que se ha encontrado, denominada mejor personal. Además, todas las partículas conocen la mejor posición encontrada dentro del enjambre $G = [g_1, g_2, \ldots, g_N]$, denominada mejor global. Existe otra variante en la que se definen sobre el enjambre sub-grupos de partículas, posiblemente solapados, a los que se le denominan vecindades, en tal caso las partículas también conocen la mejor posición encontrada dentro de su vecindad $L_i = [l_{i1}, l_{i2}, \ldots, l_{iN}]$, a la que se denomina "mejor local". Suponiendo el uso de la información proveniente del mejor global, en cada iteración t del algoritmo PSO, cada componente j de la velocidad y la posición de cada partícula i del enjambre se actualiza conforme a:

$$v_{ij}^{t+1} = \theta v_{ij}^{t} + C_1 rand()(p_{ij}^{t} - x_{ij}^{t}) + C_2 rand()(g_j^{t} - x_{ij}^{t})$$
$$x_{ij}^{t+1} = x_{ij}^{t} + v_{ij}^{t+1}$$

Donde identificamos a θ como el parámetro de inercia que regula el impacto de las velocidades anteriores en relación a la nueva velocidad de la partícula (Generalmente a θ se le asigna un valor fijo de 0.8 y en otros casos se le asigna un valor inicial entre 1 y 1.5 que decrece durante la ejecución del algoritmo), C_1 es el parámetro cognitivo que indica la influencia máxima de la mejor experiencia individual de la partícula en su nueva velocidad y C_2 lo definimos como el parámetro social, que indica la influencia máxima de la información social con respecto al nuevo valor de velocidad de la partícula (a los pesos C_1 y C_2 generalmente se les suele asignar el valor 2). Mientras que, $rand()$ es una función que retorna un número aleatorio en el intervalo [0, 1], mediante el cual se determina la influencia real de la informacion individual y social con respecto a la nueva velocidad para la partícula. El algoritmo general del PSO se ilustra en la figura 12:

Un problema común dentro del proceso de ejecución del algorito PSO es que las velocidades constantes de las partículas pueden aumentar o decrecer, para esto se emplean cotas para las velocidades. Para determinar los valores máximos y mínimos se utiliza $[-V_(max)$,$V_(max)]$. Cuando el $V_(max)$ alcanza un valor demasiado grande las partículas pueden sobrepasar e ignorar continuamente la zona con la solución global, Por otro lado, cuando $-V_(max)$ alcanza valores muy pequeños las partículas explorarán el espacio de soluciones muy lentamente y podrán quedar atrapadas alrededor de soluciones locales, incapaces de librarse de la base de atracción.

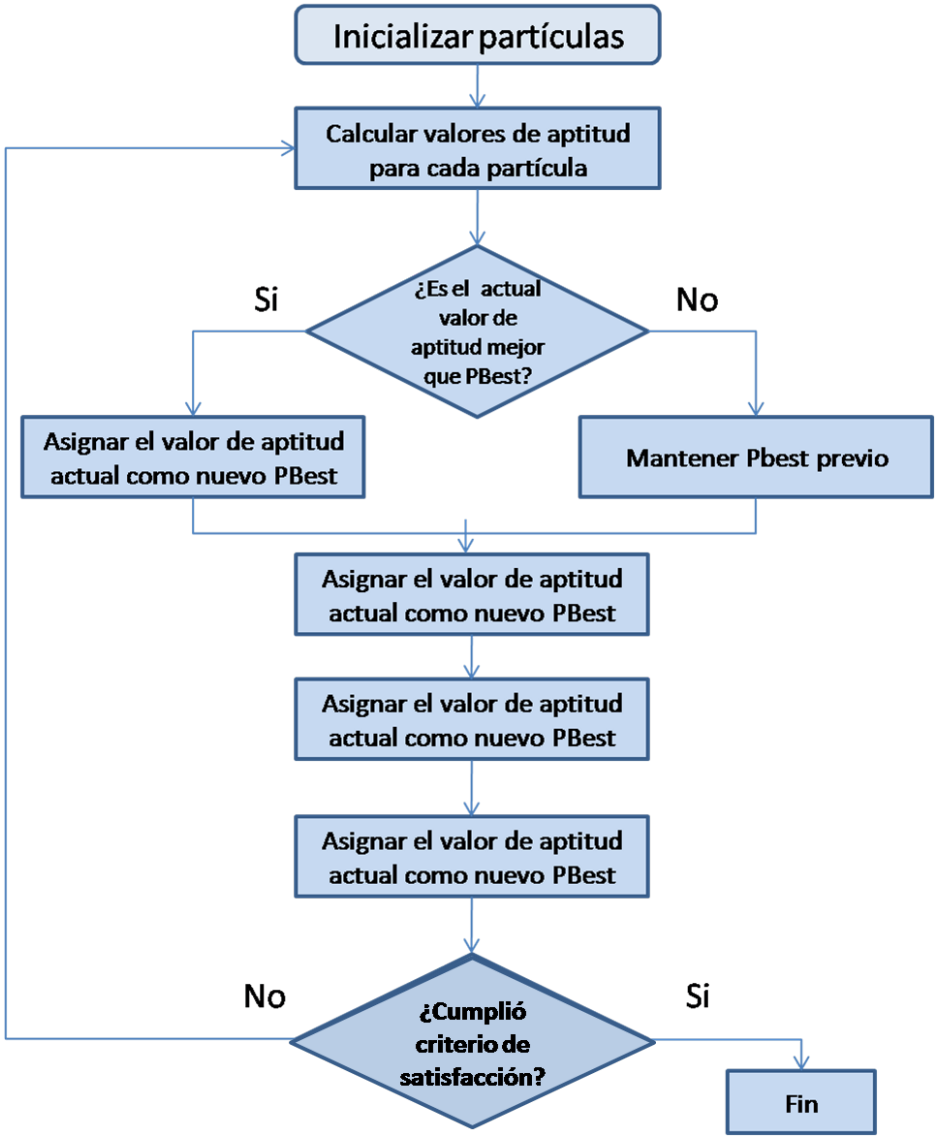

Figura 12: Algoritmo PSO.

10.1.2. QPSO (Quantum particle swarm optimization)

El desarrollo obtenido en el campo de la mecánica cuántica se debe principalmentea las conclusiones de Bohr, De Broglie, Schrödinger, Heisenberg y Bohn en el siglo XX. Sus estudios obligaron a los científicos a replantearse la aplicabilidad de la mecánica clásica y la comprensión tradicional de la naturaleza de los movimientos de objetos microscópicos.Según el PSO, una partícula es declarada por su vector de posición x_i y su vector de velocidad v_i, que determinan la trayectoria de la partícula, esta se mueve a lo largo de una trayectoria determinada por la mecánica newtoniana, sin embargo, si tenemos en cuenta la mecánica cuántica, entonces la trayectoria no tiene sentido, pues la posición x_i (notación científica) y la velocidad v_i de una partícula no pueden determinarse, por lo tanto, si las partículas individuales en un sistema PSO tienen un comportamiento cuántico, se estará muy lejos de tratar el problema de la manera clásica del PSO. Es así como definimos al QPSO como una mejora para el PSO, tratando una menor cantidad de datos de parámetros a trabajar, el rendimiento obtenido ante esta modificación en distintos tipos de problemas es concluyente, definiendo en todos los procesos una mejora en los resultados obtenidos [10]. El siguiente diagrama de flujo (figura 14) muestra el trabajo del QPSO

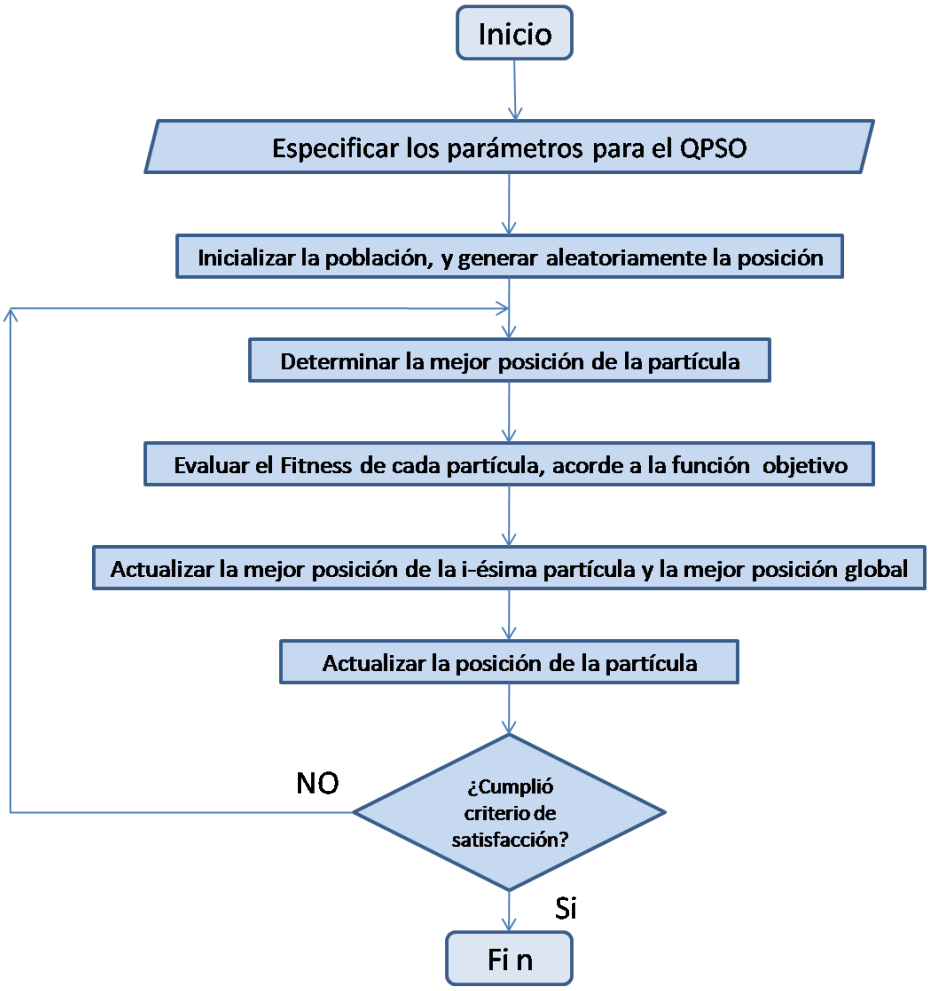

Figura 13: Diagrama QPSO.

10.2. Definición del optimizador

10.2.1. Parámetros

- V_CUTOFF = Parámetro que guarda el nombre del indicador utilizado como CutOFF, definido en los menús anteriores a la ejecución del optimizador.
- $CONDICION\_CUTOFF$ = Parámetro que almacena el valor tomado como CutOFF, definido en los menús anteriores a la ejecución del optimizador. //Fitness
- $V\_FITNESS$ = Parámetro que almacena el valor tomado como CutOFF, definido en los menús anteriores a la ejecución del optimizador. //Parámetros PSO
- $NUMERO\_PARTICULAS$ = Parámetro que almacena el valor tomado como enjambre del optimizador QPSO;
- $DIMENSION\_PARTICULA$ Parámetro que almacena el valor tomado como, este se calcula en base a la cantidad de indicadores usados.
- $PESO\_INERCIA$ = Parámetro que almacena el valor tomado como peso en el QPSO.
- $VMAX$ = Parámetro que almacena el valor tomado como la velocidad.
- $FACTOR\_COGNITIVO$ = Parámetro que almacena el valor tomado como
- $FACTOR\_SOCIAL$ = Parámetro que almacena el valor tomado como factor social.
- $NUMERO\_ITERACIONES$ = Parámetro que almacena el valor tomado como el número de iteraciones para el proceso del QPSO.

En la figura 14 se muestra como se configuran algunos parámetros con los cuales trabaja el QPSO.

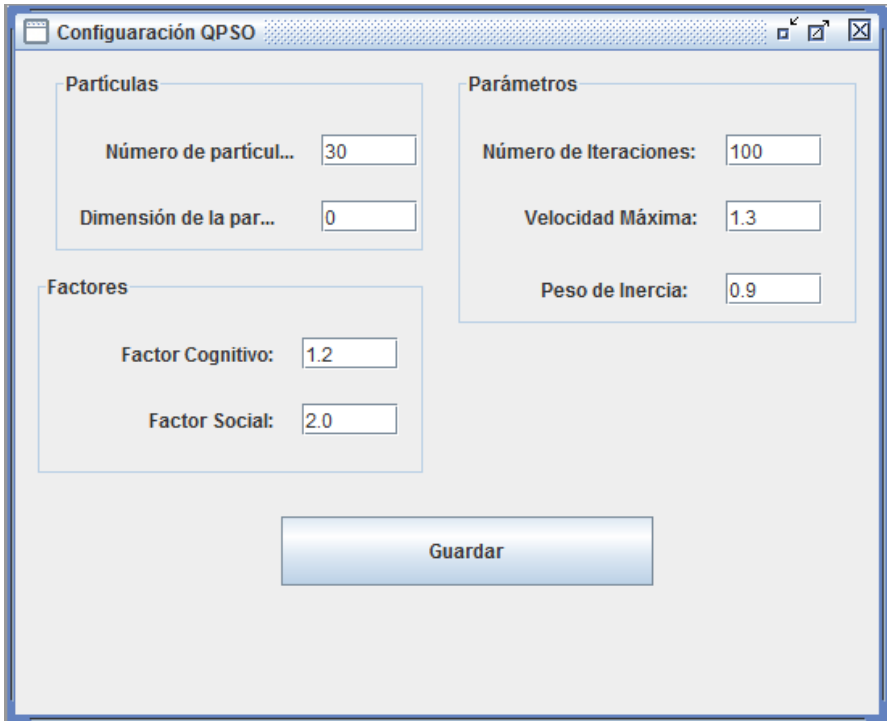

Figura 14: Configuración de parámetros QPSO.

11. Propuesta

La propuesta planteada es la de realizar un mecanismo multinivel de elección de estrategias, para lo cual se utilizará ECLiPSe para implementar el proceso de búsqueda de solución al modelo de satisfacción de restricciones. En ECLiPSe se implementarán los modelos de CSP para los distintos problemas, junto con las estrategias de búsqueda de solución a estos problemas.

ECLiPSe se comunicará con el componente desarrollado en el lenguaje de programación JAVA. Mediante la comunicación se enviarán los resultados del proceso de búsqueda y también se recibirán desde JAVA las decisiones tomadas respecto a las estrategias que se deben utilizar. Este envío de información hacia el solver (ECLiPSe) corresponderá a lo denominado como Estrategia de Enumeración y que gracias a observaciones continuas sobre el estado de la resolución del problemas, proporcionarán la información para determinar el comportamiento de la búsqueda sobre la estrategia de Enumeración que se está ejecutando. Con dicha información se podrá tomar la decisión si mantener la estrategia o bien seleccionar otra que probablemente sea más eficiente en la búsqueda de una solución. Esta información u observaciones sobre el proceso de búsqueda corresponden a lo denominado como Indicadores o bien monitores que evidencian el comportamiento de la resolución del problema.

Teniendo los Indicadores correspondientes del estado resolutivo del problema, la forma en que esta información decidirá si una estrategia es mejor que otra, será a través de una función de elección o Choice Function. En tanto la componente que corresponde al enjambre de partículas, será entrenada con una función fitness que será uno de los mismos indicadores y considerando un criterio de corte en la resolución del problema de satisfacciones en ECLiPSe. Este criterio de corte se denominará como cutoff, el cual representará cual será el tiempo o la condición de término en la búsqueda de la solución, un ejemplo de este criterio podría ser, el porcentaje de niveles alcanzado en el árbol de búsqueda, porcentaje de variables instanciadas, porcentaje de pasos alcanzados, entre otras.

Se podrá notar que el valor de cutoff intenta ser un valor representativo, independiente del problema que se desea ejecutar, esto evitará realizar cortes prematuros o demasiados extensos para un determinado problema. El objetivo del entrenamiento del enjambre es buscar la mejor partícula que indicará cuales son los mejores parámetros para la función de elección de estrategias, la cual elegirá que estrategia se adecua mejor al problema.

11.1. Conexión a eclipse

La conexión entre ECLIPSE 6.0 y Netbeans IDE 6.8, tiende a realizarse independiente del camino escogido, ya sea al elegir o no un optimizador, o entre utilizar un algoritmo genético o un algoritmo QPSO como optimizador. El caso es que siempre se ejecutan las siguientes tareas:

- Establecer la conexión con ECLIPSE 6.0. Paso mediante el cual, se lleva a cabo a través de la función eclipse.start_and_conexion() ilustrada en la imagen 15

```
public void start_and_conexion() throws Exception {

        // System.out.println("start_and_conexion");

        // Conecta el eclipse con el JVM
        eclipseEngineOptions.setUseQueues(false);
        eclipseEngineOptions.setGlobalSize(1000000);
        // Inicia el eclipse
        try {
        eclipse = EmbeddedEclipse.getInstance(eclipseEngineOptions);
        }catch (EclipseTerminatedException e) {
                System.out.println("sesion eclipse ya instanciada");
        }
}
```

Figura 15: Método de inicio y conexión.

- Establecer las variables que funcionarán como emisor y receptor desde un lado a otro y viceversa, guardando los datos respectivos de acuerdo a la situación. Paso mediante el cual, se lleva a cabo a través de la función eclipse java_setting().

```
 * @throws Exception
 */
public void  java_setting   throws Exception {
    try {
        java_to_eclipse = eclipse.getToEclipseQueue("java_to_eclipse");
        eclipse_to_java = eclipse.getFromEclipseQueue("eclipse_to_java");
        eclipse_to_java.setListener(new PSOTermConsumer());
        java_to_eclipse.setListener(new PSOTermProducer());
    } catch (EclipseException e) {
        System.out.println("Se ha producido un error al estableces " +
            "conexion con Eclipse");
        e.printStackTrace();
        throw e;
    } catch (IOException e) {
        System.out.println("Se ha producido un error al estableces" +
            " conexion con Eclipse");
        e.printStackTrace();
        throw e;
    }

}
```

Figura 16: configuración de Java.

La función trabaja buscando la mejor solución encontrada para asignar los alfas.

```
public static void alpha_setting (double[] bestsolution, PSOIndicador[] indicadores)
{
    int x = 0;
    int countInd = indicadores.length;
    for(int i = 0; i < countInd; i++){
        if(indicadores[i].isEnable()){
            indicadores[i].setAlfa(bestsolution[x]);
            x++;
        }
    }
}
```

Figura 17: configuración de alfas.

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

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

11.2. Función de elección

Como se mencionó previamente, la Choice Function, nos permite generar un ranking de las estrategias seleccionadas para la solución del problema con la ayuda de los indicadores.

La función de elección para este proyecto tendrá la siguiente forma: $CF = \sum_{i=1}^{n}(\alpha_i I_i + \beta_i H_i)$ Donde:

- n: Es la cantidad de indicadores que se desean observar.
- α_i: Es el parámetro de elección. En otras palabras es el parámetro de control sobre la importancia del indicador.
- I_i: Indicador observado desde el proceso de búsqueda de solución.
- β_i: Factor de relevancia del valor histórico del indicador I_i
- H_i: Valor histórico del indicador I_i

Como se mencionó anteriormente la finalidad del proceso QPSO será encontrar los mejores parámetros α_i, que determinarán la importancia del indicador I_i .y asi mediante el ranking mantener que estrategia es la que entrega el mejor valor.

11.3. Enjambre

Según lo estudiado anteriormente sobre PSO, el objetivo de esta investigación es crear un cúmulo de partículas, en el cual cada una de ellas represente los valores para los parámetros.

El proceso de entrenamiento del enjambre (cumulo de partículas) se realizará mediante la resolución de una pequeña porción del problema. Cuando termine el proceso de entrenamiento de la partícula, se ejecutará el proceso de resolución del problema de forma completa, seleccionando aquella partícula que tenga el mejor fitness.

11.4. Función fitness del enjambre

Anteriormente se denotó que las partículas necesitan una función fitness que evaluará a cada una de las partículas durante el proceso de entrenamiento. Esta función fitness determinará los valores de los parámetros sobre la Choise Function antes descrita y va a estar determinada por los Indicadores que fueron probados como evaluadores.

La función fitness va a estar dada por la siguiente ecuación: $fitness = f(I_i)$ Donde:

- $f(I_i)$: Será la ecuación de cálculo para el indicador $I_i, \forall i$

11.5. Flujo de implementación

La siguiente imagen muestra el flujo general de la implementación para la configuración de los parámetros de la función de elección según un enjambre de partículas.

Como se mencionó anteriormente, por cada partícula existente en el enjambre se realizará una ejecución del problema seleccionado, de tal manera que durante éste proceso los valores de los Indicadores sean analizados para tener una visualización del estado de la búsqueda. Cada partícula evaluará su mejor posición según la función fitness, permitiendo entregar los parámetros necesarios para el cálculo de la Función de Elección de estrategias. Una vez que se ejecute la Función de Elección de Estrategia se volverá a ejecutar el problema, esta vez con la siguiente partícula del enjambre, esto se realizará hasta que la totalidad de las partículas sean evaluadas y entrenadas. Cuando se complete la totalidad de las iteraciones para el entrenamiento del enjambre se elegirá la partícula que tenga la mejor posición global del enjambre, asignando los valores de su posición a los parámetros α_i de cada indicador de la Función de elección de estrategia. Finalmente se ejecutará el proceso completo de búsqueda para obtener el resultado general del problema.

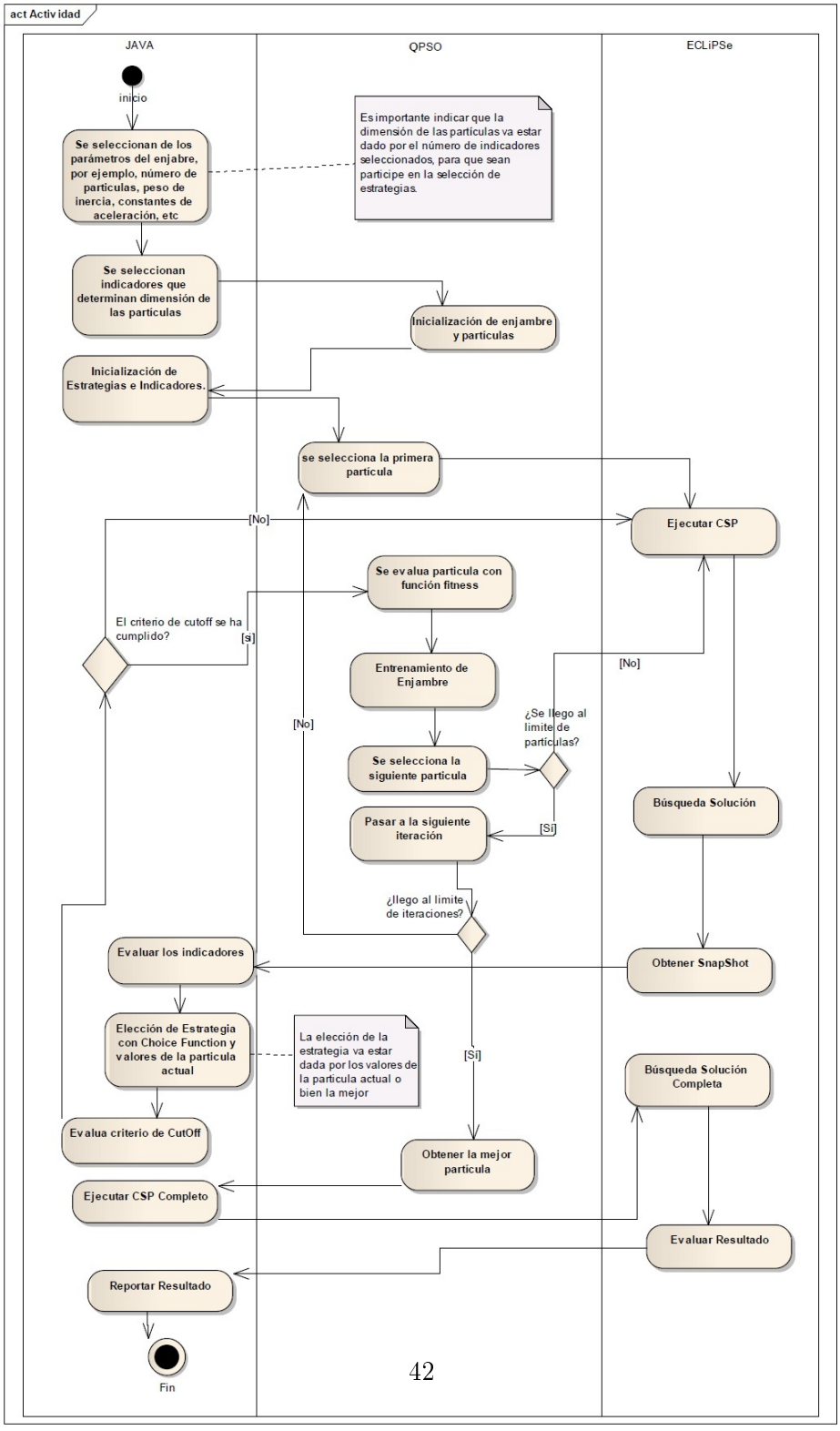

Figura 18: Flujo de implementación para la búsqueda de parámetros a través de PSO.

12. Experimentos y pruebas

En la siguiente sección se expondrán las pruebas para medir el funcionamiento del Algoritmo PSO y QPSO sobre algunos Problemas de satisfacción de restricciones, principalmente N-queens. Las pruebas sobre el Algoritmo PSO servirán como un medio para comparar y determinar si en la práctica la utilización de QPSO es ventajosa. Es necesario tener en cuenta, también, el desempeño de las estrategias utilizadas para la resolución del problema. Se realizarán pruebas unitarias a las estrategias de enumeración para su posterior análisis.

12.1. Configuración de parámetros QPSO

En la siguiente tabla de experimentos se mostrará qué indicador presenta el fitness con el cual el QPSO presenta los mejores resultados, basándonos en el número de Backtrack y el runtime promedio que se obtuvo al realizar las pruebas. El criterio de cutoff que se utilizará para realizar las pruebas será "D" (Deep) el cual trabaja la profundidad del árbol de búsqueda, es decir el indicador además de indicar la profundidad en el árbol de búsqueda nos muestra la cantidad de variables instanciadas hasta ese momento, es equivalente al valor VF (número de variables instanciadas). El valor de criterio de corte se escogerá de manera tal que no realice un porcentaje de búsqueda muy alto y tampoco un porcentaje muy pequeño. Después de analizar los resultados obtenidos en las pruebas (figura fitness) el indicador escogido para utilizar como fitness será el IN3 (Reducción del espacio de búsqueda) a pesar de que no es el primero de la lista de los mejores fitness, es el que mantiene un comportamiento más regular en las pruebas realizadas. Es necesario tener en cuenta que el tiempo (RunTime) está indicado en milisegundos. El problema escogido para resolver será el N-Reinas con n=12.

Indicadores Fitness	Max/Min	RunTime	BackTrack
Step	Min	2765	28,2
N	Min	2657,4	20,4
VU	Min	2652,2	25,6
SB	Min	2726,6	16,2
B	Min	2662,6	22
VF	Max	2662,2	23,4
VFES	Max	3068,86	20,4
TAVFES	Max	3185,6	20
PTAVFES	Max	3243,8	22,4
PVFPS	Max	2999,2	22
TAVFPS	Max	3092,6	24,8
PVFPS	Max	3012,6	21
PTAVFPS	Max	2877	18,6
TSB	Min	3172,8	23,4
D	Max	2799,2	23,4
DMAX	Max	2742,6	16,8
IN3	Max	2767,2	18,4
DB	Min	2753,8	24,8
IN1	Max	2724,2	18,6
IN2	Max	2736	22
PVFES	Max	2729,6	19,8

Tabla 2: Resultados búsqueda de Indicador Fitness (10 ejecuciones por caso)

Iteraciones	Partículas	Peso de inercia	BackTrack	RunTime	Steps
125	35	0,95	25	3420	49,75
125	30	0,95	22,3	4160,4	46,3
125	25	0,95	18,67	3316	41,34
100	35	0,95	20,75	4074,5	45,25
100	30	0,95	22	3310	46,5
100	25	0,95	17,5	2724,25	40,5
75	35	0,95	22,75	2916,25	47,25
75	30	0,95	24,5	2511,75	46,5
75	25	0,95	24	2215,25	49,75
125	35	0,9	21	4598,5	45,5
125	30	0,9	22,5	4718	47,75
125	25	0,9	20	4056,5	44,75
100	35	0,9	22,75	4158,75	47,75
100	30	0,9	19	3504	42
100	25	0,9	20	2963	43,5
75	35	0,9	27	3230,5	54,25
75	30	0,9	22,25	2725,75	51
75	25	0,9	20,75	2464,5	42,25
125	35	0,8	17,25	5407,75	41,25
125	30	0,8	17,5	4700,75	41,75
125	25	0,8	23	4290,5	37,25
100	35	0,8	19	4489,5	42,75
100	30	0,8	23,25	3785,75	48,25
100	25	0,8	22	2762	46,5
75	35	0,8	22,5	2935,5	48,25
75	30	0,8	20,25	2517,25	45,75
75	25	0,8	18	2142	44

Tabla 3: Resultados configuración QPSO - (10 ejecuciones por caso)

De la tabla anterior se desprende que la mejor configuración en base a un comportamiento promedio similar entre los resultados obtenidos y en base a los resultados entregados analizando el runtime, el backtrack y los step se determinó que la mejor configuración del QPSO es de 100 iteraciones 25 partículas y un peso de inercia de 0.95.

12.2. Desarrollo de las pruebas

El criterio de selección de las pruebas a evaluar y los indicadores utilizados para observar el desempeño de las estrategias es en base a la fácil interpretación y visualización que estos problemas e indicadores entregan. Para la realización de las pruebas utilizando el algoritmo QPSO se consideraron los siguientes CSP:

- N-reinas.
- Magic Square.
- Knight Tour.
- Sudoku.

Los indicadores escogidos fueron los siguientes:

- Número de vueltas atrás (Backtrack)
- Nodos visitados
- Tiempo de ejecución (RunTime)

12.2.1. Pruebas sobre estrategias

Se realizaron pruebas a las estrategias que se utilizarán dentro del proceso de búsqueda con los algoritmos PSO Y QPSO, resolviendo el problema de las N-reinas con N = 8, 10 12, 15 y 20. El objetivo de estas pruebas fue determinar si existen estrategias que presentan mejor funcionamiento para la naturaleza de un problema en particular, y de ser así, cuales son estas. La tabla que se muestra a continuación expone las pruebas realizadas a cada una de las estrategias de enumeración para determinar cuáles son las que muestran mejor funcionamiento:

	8			10			12			15			20		
Estrategias	Nodo	BT	Step	Nodos	BT	Step	Nodos	BT	Step	Nodos	BT	Step	Nodos	BT	Step
1	42	10	18	39	6	15	88	15	34	383	73	143	59540	10026	22220
2	59	11	17	72	12	20	82	11	24	5618	808	1389	18353	2539	4322
3	41	10	18	21	4	12	85	16	39	15	1	11	72	11	39
4	42	10	18	39	6	15	88	15	34	383	73	143	59540	10026	22220
5	42	10	18	39	6	15	88	15	34	383	73	143	59540	10026	22220
6	59	11	17	72	12	20	82	11	24	5618	808	1389	18353	2539	4322
7	41	10	18	21	4	12	85	16	39	15	1	11	72	11	39
8	42	10	18	39	6	15	88	15	34	383	73	143	59540	10026	22220

Tabla 4: Pruebas a estrategias de enumeración

Al analizar la tabla se percibe que las estrategias 3 y 7 se comportan de mejor manera en comparación a las otras cuando el número de variables crece.

12.2.2. Pruebas QPSO

Las pruebas que se muestran a continuación corresponden a las realizadas sobre el algoritmo QPSO con el problema N-reinas, con n = 8,10, 12, 15, 20.

	8			10			12			15			20		
Prueba	Nodos	BT	RT	Nodos	BT	RT	Nodos	BT	RT	Nodos	BT	RT	Nodos	BT	RT
1	41	10	2583	47	8	3607	85	16	4673	15	1	6748	4244	769	11663
2	37	7	2082	36	6	2765	328	43	4400	15	1	5867	10193	1413	14079
3	44	9	2157	47	8	2730	168	28	3947	3206	470	5726	72	11	9406
4	37	7	2116	45	9	2730	132	26	3412	2086	371	6139	72	11	9087
5	33	8	2048	47	8	2656	85	16	3376	1822	330	6172	72	11	7908
6	37	7	2165	45	6	2654	118	23	3446	15	1	6384	10193	1413	11071
7	37	7	2251	32	5	2620	168	28	3697	15	1	5862	4558	793	9490
8	37	7	2296	30	7	3209	85	16	4713	15	1	5879	72	11	8869
9	41	10	2302	41	7	3335	95	17	4209	15	1	5957	121	17	8599
10	31	7	2919	28	5	3374	132	21	4021	15	1	5818	1428	302	8137

Tabla 5: Pruebas QPSO N-reinas (10 ejecuciones por caso)

Se observa claramente que mientras aumenta la cantidad de variables, los tiempos de ejecución, el número de Backtracks y nodos visitados aumentan. Nuevamente se observó en el proceso de búsqueda que la tendencia era a escoger la estrategia 3 o 7, lo que determinaría que para obtener los resultados más eficientes la herramienta define como mejores estrategias de resolución a la estrategia MRV + Indomain y a la estrategia MRV + Indomain Max.

A continuacón se muetra la tabla resultante de los experimentos Para el problema N-reinas con 50 y 75 variables. Los resultados estan expresados como el promedio de los valores entregados tras 10 ejecuciones de cada problema.

Prueba	Backtrack	Nodos	RunTime
N-reina 50	920,8	5695,5	76200,1
Nreina 75	25412,2	7156931,2	823713,4

Tabla 6: Pruebas QPSO N-reinas 50 y 75(10 ejecuciones por caso)

Esta tabla muestra el desempeño del solver sobre las pruebas realizadas con sudoku 2,magic square 4, knight tour 5

	Sudoku 2			Magic 4			Knight 5		
Prueba	Nodos	BT	RT	Nodos	BT	RT	Nodos	BT	RT
1	214	28	3182	1209	122	8418	14742	767	25343
2	63	9	3074	1209	122	7447	7224	427	30093
3	64	9	3416	1209	122	6265	7868	470	35630
4	63	9	2905	293	19	7817	27138	1854	38953
5	1493	223	2989	1209	122	7050	17472	767	19911
6	63	9	2739	374	28	7925	14742	767	24758
7	167998	14329	43845	370	30	6103	14742	767	24196
8	123	28	3528	355	26	6910	14724	767	24913
9	167998	14329	47406	1209	122	11778	14724	767	16612
10	63	9	4010	1762	163	6465	23195	1243	25334

Tabla 7: Pruebas QPSO Sudoku, Magic Square y Knight Tour

Se observa una relativa similitud en cada una de las pruebas realizadas, salvo algunas excepciones. Las pruebas que requieren más variables tardan más tiempo en realizarse y muestran un mayor número de Backtracks, en algunos casos en el desarrollo de las pruebas las estrategias escogidas no eran siempre las más óptimas lo que ocasionaba que el tiempo de espera para llevar a cabo los experimentos se extendiese considerablemente.

12.2.3. Pruebas de estrategias

Las tablas que se presentan a continuación muestran los datos extraídos de pruebas realizadas con distintas estrategias de enumeración, incluyendo el método random y el optimizador QPSO como métodos de resolución.

La siguiente tabla muestra los mejores resultados, de entre 10 ejecuciones en cada estrategia, de Backtracks resultantes en el proceso de búsqueda de una solución para el problema N-reinas, con 8, 10, 12, 15, 20, 50 y 75 Variables.

Estrategia	NQ (n=8)	NQ(n=10)	NQ(n=12)	NQ(n=15)	NQ(n=20)	NQ(n=50)	NQ(n=75)
F + ID	10	6	15	73	10026	>27406	>27979
AMRV + ID	11	12	11	808	2539	>39232	>36672
MRV + ID	10	4	16	1	11	117	818
O + ID	10	6	15	73	10026	>26405	>26323
F + IDM	10	6	15	73	10026	27406	26979
AMRD +IDM	11	12	11	808	2539	>39232	>36672
MRV +IDM	10	4	16	1	11	117	818
O+IDM	10	6	15	73	10026	>26405	>26323
Random	5	8	18	98	32	>32718	>32973
QPSO	7	5	16	11	11	252	9255

Tabla 8: Número de Backtracks resolviendo diferentes instancias del problema N-Queens Con diversas estrategias.

A continuación se exponen los resultados de los mejores Backtracks encontrados utilizando la misma configuración explicada anteriormente, pero aplicada a los problemas MS (n=4), MS (n=5), Sudoku, Knight (n=5) y knight (n=6).

Estrategia	MS (n=4)	MS (n=5)	Sudoku	Knight(n=5)	Knight(n=6))
F + ID	12	910	18	767	>19819
AMRV + ID	1191	>46675	10439	>42889	>43098
MRV + ID	3	185	4	767	>19818
O + ID	10	5231	18	>18838	>19716
F + IDM	51	>46299	2	767	>19818
AMRD +IDM	42	>44157	6541	>42889	>43098
MRV +IDM	97	>29416	9	767	19818
O+IDM	29	>21847	2	>18840	>19716
Random	17	>39742	250	>40022	>35336
QPSO	19	74083	826	667	29608

Tabla 9: Backtracs EQ-10, Magic Squares, Sudoku y Knight con diferentes estrategias.

Para las pruebas sobre N-reinas presentadas anteriormente se muestran los mejores resultados de los nodos visitados en la siguiente tabla.

Estrategia	NQ (n=8)	NQ(n=10)	NQ(n=12)	NQ(n=15)	NQ(n=20)	NQ(n=50)	NQ(n=75)
F + ID	24	19	43	166	23893	>65535	>65535
AMRV + ID	21	25	30	1395	4331	>65535	>65535
MRV + ID	25	16	45	17	51	591	2345
O + ID	25	19	46	169	24308	>65535	>65535
F + IDM	24	19	43	166	23893	>65535	>65535
AMRD +IDM	21	25	30	1395	4331	>65535	>65535
MRV +IDM	25	16	45	17	51	591	2345
O+IDM	15	19	46	169	24308	>65536	>65535
Random	15	23	41	205	78	>65535	>65535
QPSO	30	21	65	58	72	1660	58799

Tabla 10: Nodos visitados problema N-Queens Con diversas estrategias.

Se presentan los mejores resultados para el número de nodos visitados, resolviendo los problemas EQ-10, Magic Squares, Sudoku y Knight.

Estrategia	MS (n=4)	MS (n=5)	Sudoku	Knight(n=5)	Knight(n=6))
F + ID	37	1901	158	3113	>65535
AMRV + ID	1826	>65535	30139	>65535	>65535
MRV + ID	22	546	76	3113	>65535
O + ID	31	13364	196	>65535	>65535
F + IDM	110	>65535	58	3113	>65535
AMRD +IDM	69	>65535	19550	>65535	>65535
MRV +IDM	230	>65535	153	3113	>65535
O+IDM	61	>65535	62	>65535	>65535
Random	47	>65535	1019	>65535	>65535
QPSO	292	836152	4871	11660	546801

Tabla 11: Nodos visitados EQ-10, Magic Squares, Sudoku y Knight con diferentes estrategias.

Registro de los tiempos de ejecución expresados en segundos para los problemas de N-reinas. El término t.o se refiere a que el tiempo límite para la ejecución se superó (time out).

Estrategia	NQ (n=8)	NQ(n=10)	NQ(n=12)	NQ(n=15)	NQ(n=20)	NQ(n=50)	NQ(n=75)
F + ID	0	0	0.031	0.109	23.468	t.o	t.o
AMRV + ID	0	0	0.015	1.625	8.391	t.o	t.o
MRV + ID	0.016	0	0.016	0.015	0.031	1.031	8.562
O + ID	0.016	0	0.015	0.109	23.109	t.o	t.o
F + IDM	0	0.015	0.016	0.109	22.922	t.o	t.o
AMRD +IDM	0	0.015	0.015	1.609	8.328	t.o	t.o
MRV +IDM	0.016	0.015	0.015	0	0.031	1.301	8.579
O+IDM	0	0	0.016	0.094	22.875	t.o	t.o
Random	0.156	0.141	0.125	0.296	2.293	t.0	t.0
QPSO	1.342	1.845	3.192	4.096	6.605	96.919	450.599

Tabla 12: Runtime N-Queens con diferentes estrategias

Finalmente se registran los tiempos obtenidos resolviendo los problemas Magic Squares, Sudoku y Knight.

Estrategia	MS (n=4)	MS (n=5)	Sudoku	Knight(n=5)	Knight(n=6))
F + ID	0.015	2.437	0.063	2.985	t.o
AMRV + ID	3.781	t.o	34.735	t.o	t.o
MRV + ID	0.015	0.516	0.016	2.61	t.o
O + ID	0.046	9.344	0.11	t.o	t.o
F + IDM	0.141	t.o	0.015	2.578	t.o
AMRD +IDM	0.062	t.o	22.953	t.o	t.o
MRV +IDM	0.0156	t.o	0.063	2.594	t.o
O+IDM	0.063	t.o	0.015	t.o	t.o
Random	0.265	t.0	0.280	t.0	t.0
QPSO	3.476	165.636	8.486	162.88	102.562

Tabla 13: Runtime EQ-10, Magic Squares, Sudoku y Knight.

12.3. Mejores alfas

Las siguientes tablas muestran los mejores valores de alfa caluclados por el algoritmo optimizador para ser utilizados por la choice function en la realización de las pruebas con N-reinas, Sudoku, Magic square, y knight tour.

La tabla expone los resultados de los mejores alfas obtenidos para el problema N-reinas, con n = 8, 10 12, 15, 20, 50 y 75. Los términos Node, TV t VU se refieren a los indicadores Nodos, Número total de Variables y Número de variables no instanciadas, en todas las pruebas realizadas se definió la configuaración de los indicadores node-TV-VU para la choice function.

Indicador	NQ (n=8)	NQ(n=10)	NQ(n=12)	NQ(n=15)	NQ(n=20)	NQ(n=50)	NQ(n=75)
NODE	85,50416	-28,56151	-5,77211	-94,75945	-34,15023	-6,17226	-45,55026
TV	-73,35176	9,66339	21,31375	-4,67885	92,48428	66,91093	71,13952
VU	38,91520	79,55517	48,78373	23,85410	4,41454	27,31916	-64,61747

Tabla 14: Mejores alfas N-reinas.

La siguiente tabla corresponde a los mejores alfas calculados para los problemas Magic Square, con n = 4 y 5, Knight tour(Kina), con n = 5 y 6 y el problema Sudoku, con n = 6.

Indicador	Magic Square		Kina		sudoku
	n=4	n=5	n=5	n=6	n=6
NODE	17,63425	44,98891	95,54912	-100	60,37186
TV	53,06369	-79,28445	-95,00557	39,78389	17,65122
VU	-100	-33,09528	16,28990	18,45969	89,70424

Tabla 15: Mejores alfas Magic Square, Kina y sudoku.

12.4. Análisis de resultados

Los resultados obtenidos en las pruebas sobre el problema N-reinas utilizando QPSO han demostrado que al operar con pocas variables no se perciben diferencias significativas, no así en pruebas con un mayor número de variables el desempeño del algoritmo QPSO es claramente superior, esto se debe a los mencionado en capítulos anteriores, el actuar de la partícula en cada caso, es decir el uso de velocidades para determinar las posiciones en el caso del PSO y en el caso del QPSO el movimiento cuántico para determinar la posición de estas.

N-Reinas		n=8	n=10	n=12	n=15	n=20
PSO	Nodos	37,5	39,8	139,6	721,9	3102,5
	Backtracks	7,9	6,9	23,4	117,8	475,1
	Runtime	2291,9	2968	3989,4	6055,2	9830,9
QPSO	Nodos	41,4	38	87,9	922,5	38555,6
	Backtracks	8,6	6,7	14,9	145,1	6240,9
	Runtime	2094,1	2888,2	3506	4610,6	17263,7

Tabla 16: Resultados promedios entre QPSO y PSO utilizando el CSP N-reinas (10 experimentos).

En el cuadro como ya mencionamos se observa un incremento exponencial a la hora de hacer comparaciones de rendimiento en las herramientas, si bien los primeros resultados resultan ser muy similares la última columna del

PSO muestra un aumento considerable en sus resultados en relación a su competidor, lo cual nos dice que para espacios pequeños de búsqueda ambos algoritmos resultan ser muy similares pero ya cuando se trabaja en espacios mucho más grandes el algoritmo QPSO tiende a ser más efectivo.

Las recopilaciones de todas las pruebas efectuadas con el algoritmo QPSO indican una homogeneidad en cuanto a eficiencia en el proceso de búsqueda, salvo algunos casos donde las estrategias seleccionadas no eran óptimas y eso se representa en valores más elevados de sus indicadores. En algunos casos fue imposible obtener resultados por lo que se optó a eliminar la estrategia AMRV + Indomain y la estrategia AMRV + Indomain Max, (estas estrategias mostraban los peores resultados individuales) ya que al no hacerlo el algoritmo nunca llegaba a una solución, después de eliminarlas las pruebas imposibles de conseguir pudieron ejecutarse sin ningún problema, aun así las pruebas con un espacio de búsqueda relativamente grandes continuaron siendo las más difíciles de obtener ya que cada prueba tenía tiempos de ejecución relativamente altos, por consiguiente se deduce que el algoritmo QPSO permite un óptimo desarrollo de resultados en espacios de búsqueda pequeños pero al trabajar con espacios más grandes el algoritmo tiende a ser menos eficiente, la desviación estándar (SD) nos da un mejor detalle de cómo se encuentra la dispersión de los resultados obtenidos observando que en muestras grandes la dispersión se dispara exponencialmente. En los cuadros siguientes observamos en detalle los datos de las pruebas realizadas con el algoritmo QPSO rescatando el mejor resultado, el peor resultado, el promedio y la desviación estándar en base a las pruebas realizados.

CSP	NQ (N=8)	NQ (N=10)	NQ (N=12)	NQ (N=15)	NQ (N=20)	NQ (N=50)	NQ (N=75)
Mejor	7	4	16	1	11	232	9255
Peor	10	9	28	230	519	4232	82979
Promedio	9	6,7	23,8	85,7	246,3	920,8	25490,2
SD	1,1547	2,4966	4,3665	105,1655	252,5668	1262,653	30341,88

Tabla 17: Resultados Backtracks QPSO N-Reinas

CSP	MS (n=4)	MS (n=5)	Sudoku	Knight (n=5)	Knight (n=6)
Mejor	19	74083	826	667	29608
Peor	166	503351	3180	1913	101846
Promedio	101	248451,6	2125,4	1015,3	71798,3
SD	50,6491	138952,7	946,0393	476,0061	34155,77

Tabla 18: Resultados Backtracks QPSO Magic Square, Sudoku, Knight Tour

CSP	NQ (N=8)	NQ (N=10)	NQ (N=12)	NQ (N=15)	NQ (N=20)	NQ (N=50)	NQ (N=75)
Mejor	30	21	85	15	72	1575	58799
Peor	42	44	137	1203	2890	24481	504215
Promedio	38,2	33,1	119,9	437,9	1433,5	5695,5	156931,2
SD	4,2895	10,6713	19,727	530,8939	1298,006	7302,375	183267,6

Tabla 19: Resultados Nodos visitados QPSO N-Reinas

CSP	MS (n=4)	MS (n=5)	Sudoku	Knight (n=5)	Knight (n=6)
Mejor	275	836152	4871	11660	546801
Peor	1761	5463130	20195	29675	1749215
Promedio	1060,8	2669892	13390,2	1249652	13390,2
SD	525,9981	1372160	6187,718	6705,959	566236,7

Tabla 20: Resultados Nodos visitados QPSO Magic Square, Sudoku, Knight Tour

CSP	NQ (N=8)	NQ (N=10)	NQ (N=12)	NQ (N=15)	NQ (N=20)	NQ (N=50)	NQ (N=75)
Mejor	1317	1845	2815	3951	6605	66535	413709
Peor	1536	1944	3052	4255	7727	96919	2261900
Promedio	1369,5	1880,7	2927,7	4104,8	7062,6	76200,1	823713,4
SD	63,1492	29,0748	76,6681	95,6844	379,1969	9560,11	758724,9

Tabla 21: Resultados Nodos Runtime QPSO N-Reinas

CSP	MS (n=4)	MS (n=5)	Sudoku	Knight (n=5)	Knight (n=6)
Mejor	3429	165636	8479	16288	102562
Peor	6390	1068212	15492	19087	302815
Promedio	4485,2	540449,7	12327,6	17597,1	215579,1
SD	1126,695	273045,6	2786,552	893,1745	92031,18

Tabla 22: Resultados Runtime visitados QPSO Magic Square, Sudoku, Knight Tour

En el análisis de las pruebas utilizando diversas estrategias se puede observar que el crecimiento exponencial de Backtracks y nodos visitados con respecto a la cantidad de variables en la totalidad de los problemas es menor en comparación a las otras estrategias observadas, aún así se mantiene en los márgenes, con respecto a lo observable en cuanto a tiempo de ejecución se aprecia un tiempo mayor en las pruebas con pocas variables. Cabe mencionar que si bien los tiempos señalados en las pruebas representan una referencia importante dentro de los datos recopilados para su análisis, su precisión es relativa debido a que cualquier proceso externo del equipo donde se realizan las pruebas puede afectar el resultado final haciendo difusa su observación, especialmente en pruebas con pocas variables ya que la diferencia en los tiempos es menos notoria.

13. Conclusiones

Durante el proceso de investigación que se ha realizado se analizaron detalladamente las características y componentes que involucran el proceso de resolución de problemas de satisfacción de restricciones. El centrarse en el estudio de la teoría del contenido propone una ventaja al momento de manipular los componentes presentes en el framework con el cual se trabajará posteriormente.

La programación con restricciones resulta ser una herramienta muy potente para el modelado de problemas de alta complejidad de diversas áreas y muchas veces cotidianos como problemas de asignación de turnos laborales (calendarización), planificación, investigación de operaciones, etc. Estos problemas, sea cual sea su índole, son expresados mediante un conjunto de restricciones que deben ser satisfechas por las variables que son instanciadas para llegar a una solución válida.

La cantidad de lenguajes que actualmente soportan la programación con restricciones es considerable, comenzando por el lenguaje Prolog, primer lenguaje donde se incorpora la programación con restricciones, hecho por el cual se denominaba en un comienzo como programación lógica con restricciones, otros ejemplos son B-Prolog, Mozart, ECLiPSe, GNU Prolog, etc. Gracias a que existe una gran cantidad de información sobre eclipse es que se ha podido llegar a entender de una forma mucho más clara que otros Solvers lo que ha permitido un avance incremental en el desarrollo del proyecto.

El sistema Autonomous Search proporciona la estrategia de trabajo de la que esta investigación se basa, implementando en su sistema el proceso de selección de heurísticas. El framework se encarga de evaluar en cierta medida cuál es la estrategia que promete mejores resultados, dentro de este proceso la Choice Function juega un rol muy importante.

Choice Function utiliza el algoritmo QPSO para calcular los parámetros que determinan el rendimiento de una estrategia, dada las operaciones utilizadas para rankear y elegir entre diferentes estrategias de enumeración. En orden para determinar el mejor conjunto apropiado de parámetros para la Choice Function es usando un enfoque multinivel. Estos parámetros son puestos a partir del algoritmo optimizador de partículas cuánticas (QPSO), el cual trabaja evaluando y evolucionando las diferentes combinaciones de parámetros, aliviando la tarea de parametrización manual.

El trabajo en la implementación del código del QPSO en la herramienta Autonomous Search, aunque muestra resultados satisfactorios, indica que a la herramienta aún le falta trabajo para llegar a ser eficiente y obtener los resultados esperados en todo tipo de problemas.

En relación a las pruebas realizadas se destaca que entre el optimizador PSO y el QPSO los resultados obtenidos son mucho más satisfactorios en este último, como se había previsto en la investigación realizada con anterioridad en el proyecto, si bien aún falta pulir de mejor manera la configuración de la herramienta como es el caso de los betas para las funciones de elección, que de manera directa pueden influir en los resultados de las pruebas realizadas por el optimizador, ya que como se observó en los experimentos realizados el hecho de que la herramienta tienda exponencialmente a aumentar el tiempo cuando se introducen problemas con cierta cantidad de variables percibe el hecho que aún falta más refinamiento sobre otros parámetros de configuración.

Referencias

[1] C. Castro E. Monfroy B. Crawford, R. Soto and F. Paredes. An extensible autonomous search framework for constraint programming. *Int. J. Phys. Sci*, 6(14):3369–3376, 2011.

[2] M. Montecinos C. Castro B. Crawford, R. Soto and E. Monfroy. A framework for autonomous search in the eclipse solver. *In proceedings of the 24th International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems (IEA/AIE)*.

[3] M. Montecinos C.Castro B. Crawford, R.Soto and Eric Monfroy. A framework for autonomous search in the eclipse solver. *In proceedings of the 24th International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems (IEA/AIE)*, pages 79–84, 2010.

[4] M.Salido F. Barber. Introducción a la programación con restricciones. *Revista Iberoamericana de Inteligencia Artificial*, 7, 2006.

[5] Joaquín Q. Lima M. Benjamín BaránC. *Optimizacion de Enjambre de Partículas aplicada al problema del cajero viajante bi-objetivo*. PhD thesis.

[6] M. Berlinger. *"Selector de Estrategias de Enumeración en Constraint Programming"*. PhD thesis, 2010.

[7] D. Diaz and p. Codognet. The gnu prolog system and its implementation. *Artificial Intelligence*, 2000.

[8] P. Prosser J. C. Beck and R. J. Wallace. Trying again to fail-first. in workshop on constraint solving and constraint logic programming (csclp). 3419 of Lecture. Notes in Computer Science:41–55, 2004.

[9] P. Prosser J. C. Beck and R. J. Wallace. Variable ordering heuristics show promise. *In Proceedings of the 10th International Conference on Principles and Practice of Constraint Programming (CP)*, 2833 of Lecture Notes in Computer Science:711–715, 2004.

[10] Abdesslem Layeb. A quantum inspired particle swarm algorithm for solving the maximum satisfiability problem. *International Journal of Combinatorial Optimization Problems and Informatics, México.*

[11] Alan Mackworth. Consistency in networks of relations. *Artificial Intelligence*, page 99–118, 1977.

[12] Joachim Schimpf Mark Wallace, Stefano Novello. Eclipse: A platform for constraint logic programming. Technical report, Imperial College, London, 1997.

[13] L. Granvilliers R. Chenouard and P. Sebastian. Search heuristics for constraintaided embodiment design. *AI EDAM*, 23(2):175–195, 2009.

[14] N. M. Sadeh and M.S Fox. Variable and value ordering heuristics for the job shop scheduling constraint satisfaction problem. *Artif. Intell*, 86 (1):1–41, 1996.

[15] Schulte and G. Tack. *Views and Iterators for Generic Constraint Implementations*. PhD thesis, 2006.

[16] S. F. Smith and C. Cheng. Slack-based heuristics for constraint satisfaction scheduling. *In Proceeding of the Eleventh National Conference on Artificial Inteligence (AAAI)*, pages 139–144, 1993.

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

[18] V.Bustos. Una arquitectura modular para autonomous search. *in Informe Final Proyecto para Optar al título de Ingeniero Civil en Informática*, 2012.

[19] E. Monfroy Y. Hamadi and F. Saubion. What is autonomous search?80. Technical report, 2008.

14. Anexo

14.1. Guía de instalación de la herramienta Autonomous Search

1.- Entrar en la carpeta libreria_autonomous_search, que se encuentra en la carpeta JEGAP(V2.4)

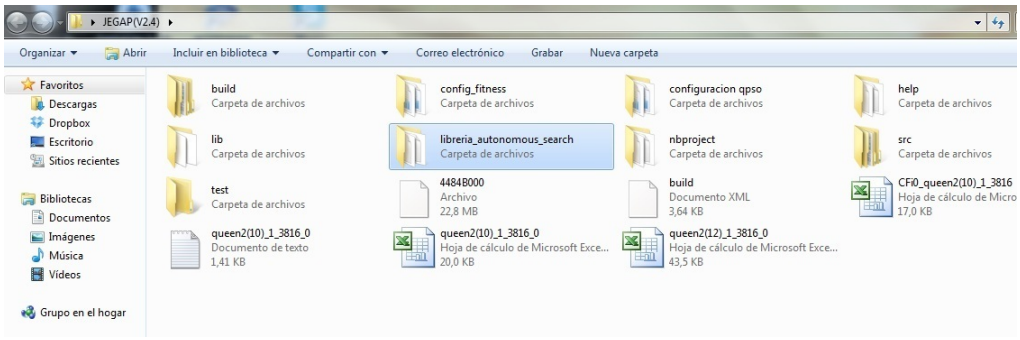

Figura 19: Carpeta Jegap(V2.4).

2.- Copiar los archivos lib_autonomous_search.eci y lib_autonomous_search.pl en la carpeta

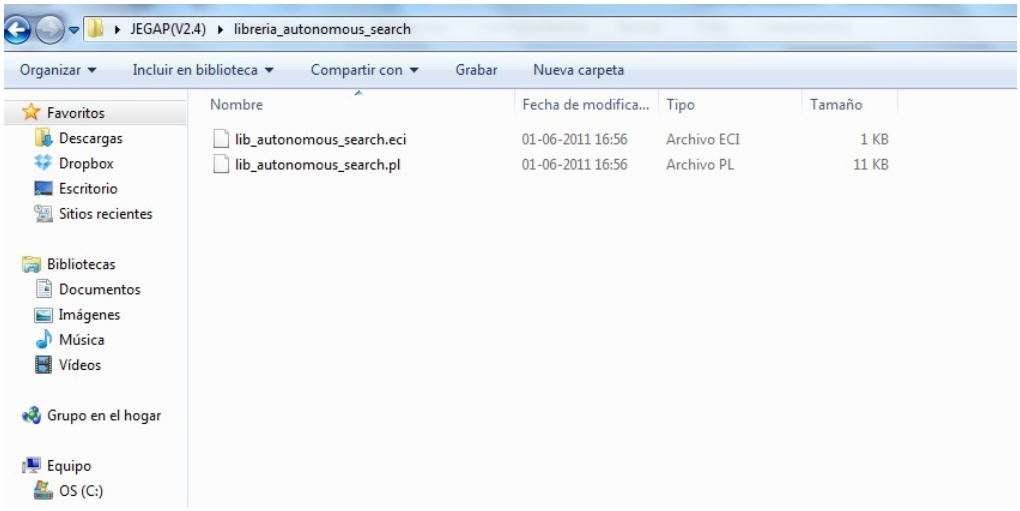

Figura 20: Carpeta librería_autonomous_search

3.- Entrar a la carpeta lib_public que se encuentra en la carpeta donde se encuentra instalado eclipse.

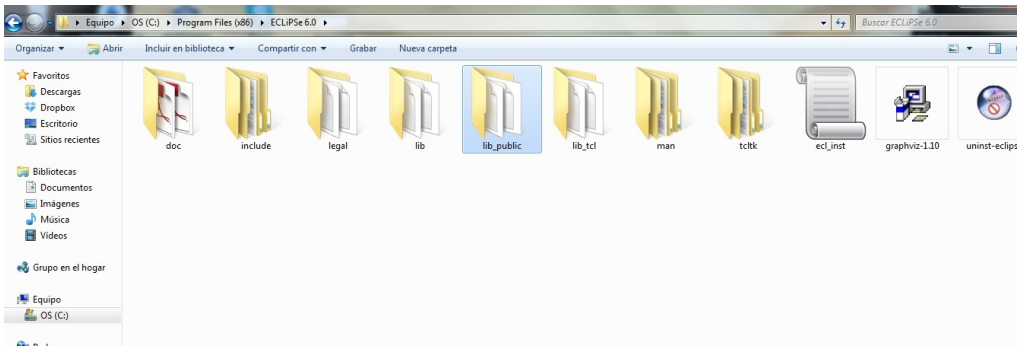

Figura 21: Carpeta ECLiPSe.

4.- Pegar los archivos copiados.

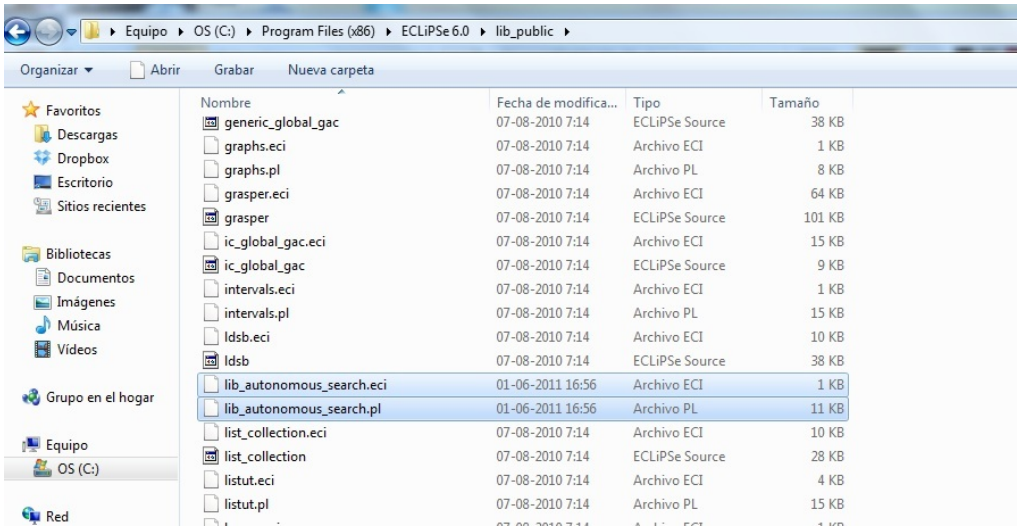

Figura 22: Carpeta carpeta lib_public.