PONTIFICIA UNIVERSIDAD CATOLICA DE VALPARAISO
FACULTAD DE INGENIERIA
ESCUELA DE INGENIERIA INFORMATICA

ENUMERACION ADAPTATIVA Y RANKEADORES DE ESTRATEGIAS BASADOS EN ALGORITMO BAT

RODRIGO OLIVARES ORDENES

INFORME FINAL PROYECTO DE TESIS
PARA OPTAR AL GRADO DE
MAGISTER EN INGENIERIA INFORMATICA

ABRIL, 2015

PONTIFICIA UNIVERSIDAD CATOLICA DE VALPARAISO
FACULTAD DE INGENIERIA
ESCUELA DE INGENIERIA INFORMATICA

ENUMERACION ADAPTATIVA Y RANKEADORES DE ESTRATEGIAS BASADOS EN ALGORITMO BAT

RODRIGO OLIVARES ORDENES

Profesor guía: **RICARDO SOTO DE GIORGIS**

Abril, 2015

Dedico este logro a mi señora Nelly Torres y a mi preciosa
hija Catalina Olivares, les agradezco el apoyo que siempre me dieron.
Agradezco a toda mi familia por el ánimo que siempre me dieron y a mis amigos,
en especial a Rodrigo Herrera que me incentivó a dar este importante paso en mi vida profesional.

Resumen

La programación con restricciones permite resolver problemas de satisfacción de restricciones y optimización, mediante la construcción y exploración de un árbol de búsqueda que contiene las posibles soluciones. Estas potenciales soluciones se generan mediante la selección de una variable y luego su valor, proceso conocido como enumeración. La estrategia de enumeración es la responsable de seleccionar el orden en que se eligen las variables y valores, para construir una potencial solución. Existen diferentes maneras de realizar esta selección, y dependiendo de la calidad de esta decisión, la eficiencia del proceso de resolución puede variar drásticamente. A partir de esto, surge la idea de gestionar esta problemática, intercalando diferentes estrategias de enumeración, en lugar de utilizar sólo una. Las estrategias son evaluadas de acuerdo a indicadores, con el fin de utilizar la más prometedora en cada parte del proceso de búsqueda. Este proceso se conoce como control en línea de las estrategias de enumeración y su correcta configuración puede ser vista como un problema de optimización. En esta tesis, presentamos un nuevo enfoque de control en línea de las estrategias de enumeración basada en la optimización Bat. El algoritmo Bat es una metaheurística relativamente nueva, basado en el comportamiento de ecolocalización que poseen ciertos murciélagos que emplean sonidos para identificar los objetos en su entorno. Se ilustran resultados prometedores, donde el algoritmo Bat es capaz de superar enfoques reportados previamente.
Palabras clave: *Programación con restricciones, algoritmo Bat, sistemas adaptativos, heurísticas, metaheurísticas, choice function.*

Abstract

Constraint programming allows to solve constraint satisfaction and optimization problems by building and then exploring a search tree of potential solutions. Potential solutions are generated by firstly selecting a variable and then a value from the given problem, process known as enumeration. The enumeration strategy is responsible for selecting the order in which those variables and values are selected to produce a potential solution. There exist different ways to perform this selection, and depending on the quality of this decision, the efficiency of the solving process may dramatically vary. A modern idea to handle this concern, is to interleave during solving time a set of different strategies instead of using a single one. The strategies are evaluated according to process indicators in order to use the most promising one on each part of the search process. This process is known as online control of enumeration strategies and its correct configuration can be seen itself as an optimization problem. In this thesis, we present a new system for online control of enumeration strategies based on bat-inspired optimization. The bat algorithm is a relatively modern metaheuristic based on the location behavior of bats that employ echoes to identify the objects in their surrounding area. We illustrate, promising results where the proposed bat algorithm is able to outperform previously reported metaheuristic-based approaches for online control of enumeration strategies.
Keywords: *Constraint programming, bat algorithm, adaptive systems, heuristics, metaheuristics, choice function.*

Índice

1. Introducción **1**

2. Definición de objetivos **3**

2.1. Objetivo general 3

2.2. Objetivos específicos 3

3. Estado del arte **4**

4. Programación con restricciones **5**

4.1. El origen de la Programación con restricciones 5

4.2. Resolución de CSP 5

4.3. Algoritmos de búsqueda 7

4.3.1. Generate and Test 8

4.3.2. Backtracking 9

4.3.3. Forward Checking 10

4.3.4. Maintaining Arc Consistency 11

4.4. Técnicas de consistencia 12

4.4.1. Nodo-Consistencia 12

4.4.2. Arco-Consistencia 13

4.5. Estrategias de enumeración 13

4.5.1. Heurísticas 14

4.5.2. Heurística de selección de variable 14

4.5.3. Heurística de selección de valor 15

5. Búsqueda Autónoma **17**

5.1. Arquitectura 17

5.2. Choice Function 19

5.2.1. Cálculo parámetro α 20

6. Algoritmo Bat **22**

7. Experimentos **25**

8. Conclusiones **39**

A. Anexos **40**

A.1. Indicadores 41

A.1.1. Lista de indicadores base 41

A.1.2. Lista de indicadores calculados 42

A.2. Modelos ECL^iPS^e 43

A.2.1. N-Queens 43

A.2.2. Sudoku puzzle 44

A.2.3. Magic Squares 45

A.2.4. Knight's Tour 46

A.2.5. Latin Squares 47

A.2.6. Langford 48

A.2.7. Quasigroup 49

Lista de tablas

1. Configuración Bat algorithm 26
2. Estrategias de enumeración 26
3. Choice functions utilizadas en los experimentos con el enfoque BAT. 27
4. Choice functions utilizadas en los experimentos sin parámetro α. 27
5. Backtracks requeridos para las diferentes estrategias de enumeración en los problemas N-Queens y Sudoku. 28
6. Backtracks requeridos para las diferentes estrategias de enumeración en los problemas Magic Square y Latin Square. 29
7. Backtracks requeridos para las diferentes estrategias de enumeración en los problemas Knight's Tour, Quasigroup y Langford. 31
8. Solving Time requerido para las diferentes estrategias de enumeración en los problemas N-Queens y Sudoku. 34
9. Solving Time requeridos para las diferentes estrategias de enumeración en los problemas Magic Square y Latin Square. 35
10. Solving Time requeridos para las diferentes estrategias de enumeración en los problemas Knight's Tour, Quasigroup y Langford. 37
11. Indicadores base 41
12. Indicadores calculados 42

Lista de figuras

1. Solución al problema 4-reinas. 6
2. Árbol de búsqueda 8
3. Algoritmo Generate and Test para 4-Queens 9
4. Algoritmo Backtracking para 4-Queens 10
5. Algoritmo Forward Checking para 4-Queens 11
6. Algoritmo Maintaining Arc Consistency para 4-Queens 12
7. Ejemplo Arco-Consistencia 14
8. Diagrama del Framework actual 18
9. Ranking de las estrategias de enumeración 20
10. Comparación backtracks. 33
11. Comparación tiempo de resolución. 38

Lista de algoritmos

1. Bat algorithm 23
2. Algoritmo para el prolema N-Reinas en ECL^iPS^e 43
3. Algoritmo para el prolema Sudoku en ECL^iPS^e 44
4. Algoritmo para el prolema Magic Squares en ECL^iPS^e. 45
5. Algoritmo para el prolema Knight Tour en ECL^iPS^e. 46
6. Algoritmo para el problema Latin Square en ECL^iPS^e. 47
7. Algoritmo para el problema Langford en ECL^iPS^e. 48
8. Algoritmo Quasi Group en ECL^iPS^e. 49

1. Introducción

La programación con restricciones es una metodología capaz de resolver problemas de satisfacción de restricciones y optimización. Bajo este marco, los problemas se formulan como un conjunto de variables y restricciones. Las variables representan las incógnitas del problema y están vinculados a un dominio no vacío de posibles valores, mientras que las restricciones definen las relaciones entre esas variables. Una solución al problema es definida por una asignación de valores a variables que no viola ninguna restricción. El proceso de resolución se realiza mediante un motor de búsqueda, comúnmente llamado *solver*, que trata de llegar a un resultado mediante la construcción y exploración de un árbol de búsqueda de las posibles soluciones.

El rendimiento y comportamiento de la mayoría de los solvers, depende en gran medida de la calidad de su configuración. En particular, la estrategia de enumeración es un componente clave de configuración que puede influir drásticamente el rendimiento de los solvers de programación con restricciones. La estrategia de enumeración es responsable de seleccionar el orden en que se eligen las variables y valores para construir las potenciales soluciones del problema. Sin embargo, la predicción a priori de la correcta estrategia de enumeración es a menudo inviable, ya que su comportamiento es comúnmente impredecible.

Una idea reciente para manejar esta problemática es la intercalación de un conjunto de diferentes estrategias a lo largo del proceso de búsqueda. De esta manera las estrategias de enumeración son reemplazadas, en función del rendimiento exhibido a lo largo de la fase de resolución. El desempeño de las estrategias se evalúa de acuerdo a ciertos indicadores, con el fin de utilizar las más prometedoras en cada parte del proceso de búsqueda.

Bajo este escenario, para lograr obtener el mejor rendimiento posible del proceso de búsqueda, Autonomous Search provee una arquitectura en la que cada uno de sus componentes posee información actualizada en todo momento. Uno de ellos posee una función de selección o choice function, encargada de priorizar cada estrategia y dada ella, seleccionar la siguiente, continuando con el proceso de búsqueda de la potencial solución.

Esta forma de búsqueda se denomina *control en línea* dentro de la búsqueda autónoma (AS) y su correcta configuración puede ser vista como un problema de optimización. De hecho, estamos en presencia de un problema de hiper-optimización que en este caso, es la optimización del proceso para resolver un problema de optimización.

En este trabajo, se plantea un nuevo enfoque para el control en línea de las estrategias de enumeración,

basada en la optimización Bat. El algoritmo Bat es una metaheurística relativamente nueva, basada en el comportamiento de la ecolocalización que poseen ciertos murciélagos que les permite identificar objetos (nidos, obstáculos, alimentos, etc) mediante la emisión de pulsos de sonido y la recuperación de los correspondientes ecos producidos. Se ilustran resultados prometedores en el que el algoritmo propuesto es capaz de superar enfoques reportados previamente, basados en metaheurísticas para el control en línea de las estrategias de enumeración.

Este trabajo se organiza de la siguiente manera: Se presenta como primera aproximación los objetivos del problema y el estado del arte, en los capítulos 2 y 3, respectivamente. Luego se entrega una descripción general de la programación con restricciones en el capítulo 4 y los conceptos asociados a la búsqueda autónoma en el capítulo 5. El problema de hiper-optimización y el algoritmo Bat se ilustran en el capítulo 6. En el capítulo 7, se presentan los experimentos y el análisis de los resultados. Por último, se plantea una conclusión y trabajos futuro.

2. Definición de objetivos

2.1. Objetivo general

Diseñar e implementar un optimizador, basado en la metaheurística del algoritmo Bat, para Autonomous Search que permita realizar un ranking de estrategias de enumeración, a partir de una choice function, para resolver problemas combinatoriales complejos.

2.2. Objetivos específicos

- Investigar y comprender el contexto de la programación con restricciones y su metodología para la resolución de problemas combinatoriales.
- Analizar y comprender los conceptos y la arquitectura de la búsqueda autónoma.
- Investigar e implementar la metaheurística del algoritmo Bat.
- Implementar nuevas choice functions (funciones de selección).
- Incrementar el portafolio de estrategias de enumeración.
- Incrementar el portafolio de problemas combinatoriales.
- Probar las nuevas choice functions, integrando las nuevas estrategias de enumeración y los nuevos problemas combinatoriales.
- Analizar los resultados.

3. Estado del arte

Autonomous Search es un enfoque relativamente nuevo que permite al solver adaptar sus propios parámetros y heurísticas, esta forma ser más eficiente sin la necesidad de contar con el conocimiento de un usuario experto. El objetivo es darle más capacidad al solver para que este pueda tomar el control y adaptar su búsqueda, basándose en algunas métricas y ajustes automáticos.

En este framework, los mecanismos de control se clasifican en dos grandes grupos dependiendo de las fuentes de información de retroalimentación: métodos online y métodos offline. Cuando la información proviene de la experiencia del entrenamiento de las instancias se le llama offline, y cuando la información es recogida en tiempo de ejecución corresponde a control online. Durante los últimos años, el uso de ambos métodos en diferentes técnicas de resolución ha presentado impresionantes mejoras al momento de resolver problemas de optimización combinatorialmente complejos. Una interesante linea de investigación en este contexto es acerca del ajuste de los parámetros en las metaheurísticas, donde diferentes enfoques han presentado buenos resultados [13, 14, 15, 16, 17, 18].

En los dominios de búsqueda completa, ambos métodos también han participado fuertemente, por ejemplo para impulsar con éxito SAT [19, 20, 21, 22] y solvers de programación entera mixta [23, 24] incluyendo ajuste automático así como la configuración adaptativa.

La integración de los métodos de control en la programación con restricciones es una línea mas reciente. En el contexto de los métodos offline, se proponen enfoques preliminares para probar y aprender buenas estrategias después de solucionar un problema, como en los trabajos reportados en [25, 26, 27]. Otra idea tras el mismo objetivo propone asociar pesos a restricciones [28], que se ven incrementados cuando se lleva acabo la búsqueda en el dominnio. De este modo, las variables que participan dentro de constantes mas pesadas pueden ser seleccionadas antes ya que son mas propensas a ocasionar fallos. Una variación de este enfoque se presenta en [29, 30], que argumentan que las decisiones iniciales son a menudo más importantes, lo que sugiere llevar a cabo una fase de muestreo a priori para obtener una mejor selección inicial.

4. Programación con restricciones

4.1. El origen de la Programación con restricciones

La programación con restricciones es el estudio de sistemas computacionales basados en restricciones, que se utilizan para describir y posteriormente resolver problemas tanto de tipo combinatorial como de optimización, por medio de la declaración de restricciones en el dominio del problema. El objetivo es encontrar soluciones que cumplan con todas estas restricciones [1, 2, 3].

Durante los años 60 a 70 comenzaron los primeros trabajos relacionados con la programación con restricción, los cuales se encontraban en el campo de la inteligencia artificial.

4.2. Resolución de CSP

La solución de un CSP consta de dos fases [3]:

- Modelar el problema como un problema de satisfacción de restricciones. Es decir modelarlo mediante un conjunto de variables, sus dominios y sus restricciones.
- Procesar el problema de satisfacción de restricción resultante. Una vez formulado el problema como un CSP, se deben aplicar dos métodos para procesarlos.
    - Técnica de consistencia. Esta técnica consiste en la eliminación de valores inconsistentes de los dominios de variable.
    - Algoritmo de búsqueda. Esta técnica consiste en la exploración sistemática del espacio de búsqueda, hasta encontrar una solución o probar que ésta no existe.

Las técnicas de consistencia permiten deducir información del problema. (Nivel de consistencia, valores posibles de variables, dominio mínimo, etc.), aunque se combinan en general con las técnicas de búsqueda, ya que la técnica de consistencia reduce el espacio de soluciones y el de búsqueda exploran dicho espacio resultante.

Si bien el modelado de los CSP's se pueden expresar de distintas formas, incluso en un lenguaje natural, la forma recomendada para la resolución es [3, 4]:

Un problema de satisfacción puede ser definido formalmente como una terna, (X, D, C), donde:

- X, corresponde a un conjunto de n variables x_1, ..., x_n

- $D = < D_1, ..., D_n >$ es un vector de dominios. La i-ésima componente D_i, es el dominio que contiene todos los valores que se le puede asignar a la variable x_i.
- C es un conjunto finito de restricciones.
- La solución para el CSP será la asignación $\{x_1 \rightarrow a_1, ..., x_n \rightarrow a_n\}$, tal que $a_i \in D_i$ para $i = 1, ..., n$ y $(a_{j1}, ..., a_{jn}) \in C_j$ para $j = 1, ..., m$.

A continuación se presentan ejemplos de CSP's con su respectivos modelado.

Ejemplo 1: N-Queens

Se requiere ubicar n reinas sobre un tablero de ajedrez de tamaño $n \times n$, de tal forma que ninguna de ellas se pueda atacar, considerando $n = 4$.

Modelo:

- Definición de las variables, en este caso serán 4 reinas: R_1, R_2, R_3, R_4.
- Se define el dominio $\{1, 4\}$, que corresponde a la cantidad de filas disponibles.
- Y el conjunto de restricciones para $i \in [1, 3]$ y $j \in [i + 1, 4]$.
  - Posicionar las reinas en distintas filas: $R_i \neq R_j$.
  - No posicionar las reinas en la diagonal noroeste, sureste: $R_i - R_j \neq j - i$.
  - No posicionar las reinas en la diagonal suroeste, noreste: $R_i - R_j \neq i - j$.

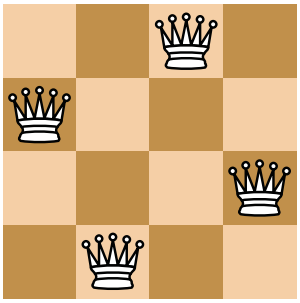

Figura 1: Solución al problema 4-reinas.

La Figura 1 muestra la disposición de 4 reinas sobre el tablero, satisfaciendo cada una de las restricciones.

Ejemplo 2: Criptoaritmática

Se requiere asignar distintos valores entre el 1 y 8 a cada letra para que la ecuación indicada sea cierta.

$$\begin{array}{cccccc} & C_1 & C_2 & C_3 & & \\ & \mathbf{I} & \mathbf{D} & \mathbf{E} & \mathbf{A} & \\ + & \mathbf{I} & \mathbf{D} & \mathbf{E} & \mathbf{A} & \\ \hline \mathbf{M} & \mathbf{E} & \mathbf{N} & \mathbf{T} & \mathbf{E} & \end{array}$$

Modelo:

- Variables, cada letra de la ecuación: **I**, **D**, **E**, **A**, **M**, **N**, **T**, C_1, C_2, C_3.
- Dominio para **I**, **D**, **E**, **A**, **M**, **N** $= [1, ..., 8]$ y para C_1, C_2, $C_3 \in \{0, 1\}$, pues son considerados las posible reservas de la suma.
- Restricciones: Corresponden a las igualdades que deben ocurrir para satisfacer la ecuacioón inicial:

$$2A = (10C_3) + E$$
$$(2E) + C_3 = (10C_2) + T$$
$$(2D) + C_2 = (10C_1) + N$$
$$(2I) + C_1 = (10M) + E$$

4.3. Algoritmos de búsqueda

Los algoritmos de búsqueda permiten, por una parte, recorrer por completo el espacio de búsqueda y localizar todas las posibles asignaciones de valores a las variables, de tal manera de poder garantizar una solución global o de lo contrario demostrar que ésta no existe [1, 3, 4]. Estos algoritmos se denominan **algoritmos completos**. Ahora bien, los **algoritmos incompletos**, exploran sólo una parte del espacio de búsqueda, por lo que su rapidez es mayor que los algoritmos completos, pero no garantizan encontrar la solución óptima global. Dada la incorporación de heurísticas en el proceso de resolución, son mayormente usados en problemas de satisfacción de restricciones como los ejemplos anteriormente nombrados y en problemas de optimización.

Como se mencionó anteriormente los algoritmos de búsqueda asignan valores a variables, es por esto que se genera un árbol de búsqueda como se muestra en la Figura 2, donde la raíz es trío (X_1, X_2, X_3) vacía,

lo que muestra que aún no se han asignado valores a las variables. Así en el primer nivel se le ha asignado valor a X_1, siguiendo esta misma lógica los próximos niveles tendrán las correspondientes asignaciones de valores a las variables representadas por X. Cabe destacar que para un árbol donde existe m cantidad de variables los nodos del nivel m, que representan también las hojas del árbol, corresponderán la asignación de todos los valores a variables contenidos en el problema y si alguna de estas asignaciones (de los nodos hoja) no transgrede ninguna restricción entonces es solución.

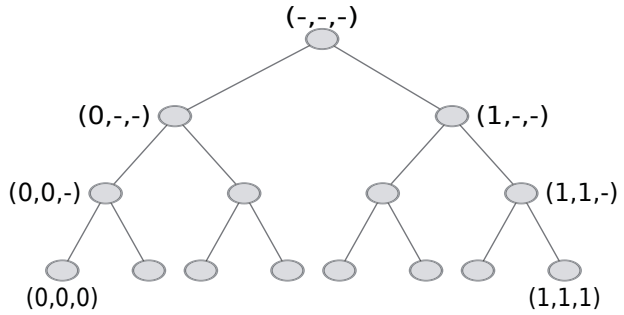

Figura 2: Árbol de búsqueda

Existen diversos algoritmos de búsqueda, de tal manaera que cada uno utiliza un método diferente en busca de una solución. Algunos pueden resultar mejores que otros en cuanto a su efectividad y/o rendimiento. A continuación se explicarán brevemente los algoritmos de búsquedas más comunes, resaltando sus características.

4.3.1. Generate and Test

Es una de las técnicas de búsqueda más sencillas, permite encontrar la solución de un problema de satisfacción de restricciones. Su funcionamiento se basa en que cada combinación posible de la asignación de variables y dominio es generada [3]. Luego se prueba si la combinación cumple con el conjunto de restricciones, la primera combinación que lo haga será la solución al problema.

Este algoritmo de búsqueda tiene como desventaja que asigna todas las variables a cada elemento del dominio asociado, por lo que si existe una gran cantidad tanto de variables como de elementos en el dominio, la cantidad de combinaciones será igual al producto cartesiano de ambos conjuntos, generando una gran cantidad de asignaciones que podrían no llevar a la solución, por lo cual se considera un algoritmo ineficiente para problemas con gran cantidad de variables y de amplios dominios [3].

La Figura 3 muestra gráficamente el algoritmo de Generate and Test para el problema de las 4 reinas descrito anteriormente, se puede ver claramente el alto costo de usarlo en este problema, ya que la gran mayoría de las primeras asignaciones completas (todas las variables han sido asignadas a un elemento del dominio) no satisfacen algunas de las restricciones y por lo tanto no son soluciones.

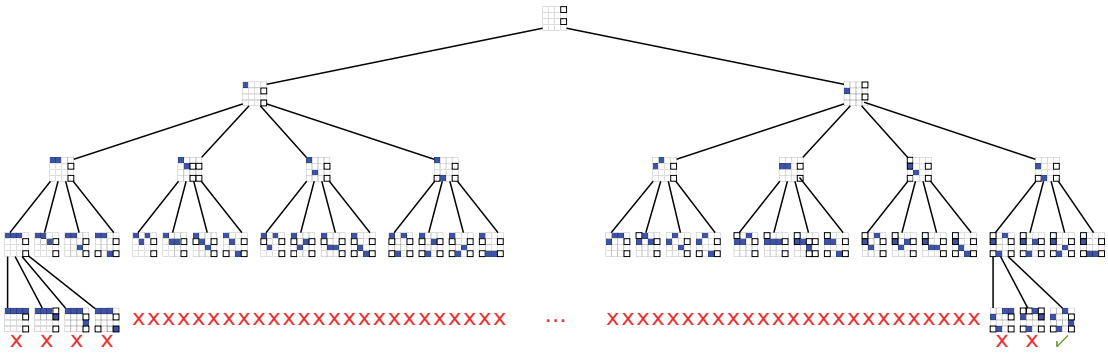

Figura 3: Algoritmo Generate and Test para 4-Queens

4.3.2. Backtracking

También llamado **Backtracking Cronológico**, es una mejora al algoritmo Generate and Test. la cual permite que cada vez que se asigna un nuevo valor a una variable actual (R_1), se comprueba si éste es consistente con los valores que hemos asignado a las variables pasadas. Si la consistencia no se cumple, se abandona esta asignación parcial y se le asigna un nuevo valor a R_1, repitiendo el chequeo de consistencia. Si ya se agotaron todos los valores de su dominio, el algoritmo Backtracking retrocede para probar otro valor para la variable R_{i-1}. Si tras asignarle otro valor y comprobando consistencia, nuevamente se agota el dominio (D_{i-1}), se repite el proceso para el nivel $i-2$ y para los niveles necesarios que posean esas mismas características [3].

En resumen la mejora que realiza Backtracking, permite detectar inconsistencias en la asignación parcial para descartarla, ya que en los siguientes subárboles no podrá existir una asignación completa que sea solución al problema, ahorrándose recorrer y verificar los subárboles que cuelgan de la asignación parcial como lo haría el algoritmo Generate and Test.

La desventaja de este algoritmo es que detecta inconsistencias que se producen por la misma razón en distintas partes de la búsqueda. Otra desventaja es que sólo comprueba inconsistencia entre la variable actual

respecto de la(s) anterior(es), pudiéndose así generar inconsistencias de la variable actual y las futuras, como lo muestra la Figura 4 en el recuadro rojo, donde la ubicación de las dos reinas es consistente, pero ninguna posición de la tercera reina será consistente con la posición de sus predecesoras.

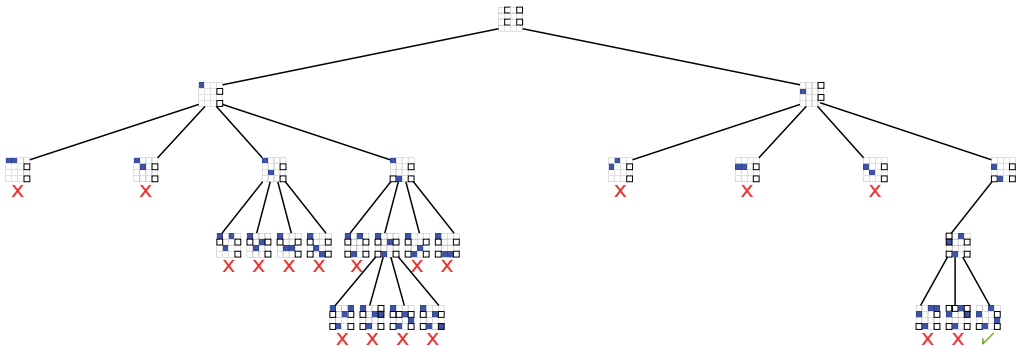

Figura 4: Algoritmo Backtracking para 4-Queens

4.3.3. Forward Checking

Es un algoritmo tipo **Look-Ahead** (comprobación inferencial hacia adelante) cuya principal característica es que revisa los valores futuros de la asignación actual, es decir que si existen inconsistencias de la variable actual y las futuras entonces esos valores no serán considerados. Además realiza el chequeo de la variable actual con las pasadas [3].

Forward Checking revisa la asignación actual con todos los valores del dominio de las futuras variables y que dada las restricciones sobre la asignación actual no podrán ser considerados. Así estos valores son borrados del dominio temporalmente y en caso de que éste quede sin valores, la variable actual debe ser utilizada con otro valor, repitiendo el proceso de chequeo de las variables futuras. Si ningún valor instanciado es solución, entonces se realiza Backtracking.

Este algoritmo limita el espacio de búsqueda, ya que al revisar las variables futuras se reduce el dominio posible para las variables que serán instanciadas posteriormente, pese a que es un algoritmo bastante más eficiente que los descritos anteriormente posee la desventaja que no evalúa la consistencia de las variables futuras con otras futuras; esto podría ser posible ya que este algoritmo conoce los valores del dominio que pueden ser instanciados y posiblemente verificar la consistencia de éstos con los futuros.

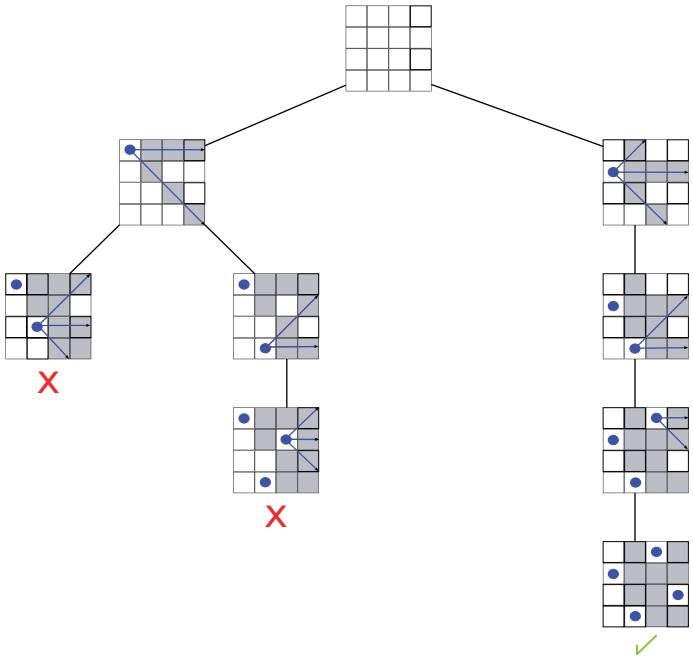

Figura 5: Algoritmo Forward Checking para 4-Queens

4.3.4. Maintaining Arc Consistency

Este algoritmo elimina inmediatamente de la primera asignación, aquello que genera un conflicto posterior. Luego, con los elementos del dominio que estén aún disponibles, realiza el mismo proceso verificando inmediatamente si existe una solución dada la primera asignación o devolviéndose, eliminando esa asignación si no existiera una solución dada esas características [3].

La Figura 6 muestra el funcionamiento de este algoritmo y en las matrices a, b, c y d muestra más detalladamente lo que ocurre. Si la primera reina (R_1) es colocada en la posición $(1, 1)$, en matriz a se ve que el dominio que genera conflicto con esa posición y las futuras es eliminado (recuadros azules). Se puede observar que aún queda dominio para las siguientes reinas, por lo tanto el algoritmo comienza a revisar esos casos y lo hace con la celda $(3, 2)$. Se descubre que esa posición, para la segunda reina (R_2) no es consistente (recuadros naranjos), pues no deja lugar para las siguientes reinas (R_3 y R_4)), así la asignación de la segunda reina en la celda $(3, 2)$ es eliminada, procedimiento que se observa en la matriz b. El algoritmo continúa con la posición $(4, 2)$ para colocar la siguiente reina (R_2), elimina los posibles conflictos las futuras posiciones (recuadros rojos en matriz c), entonces se establece que la posición para la tercera reina sería la celda $(2, 3)$ que se muestra en la matriz d, pero se provocaría inconsistencia, ya que no queda una posición

disponible para la cuarta reina (R_3)), por lo tanto para la posición de la primera reina en la celda $(1, 1)$ no existe solución. El algoritmo para otra primera reina en otra posición se repite de forma equivalente hasta llegar a la solución como se muestra en la Figura 6.

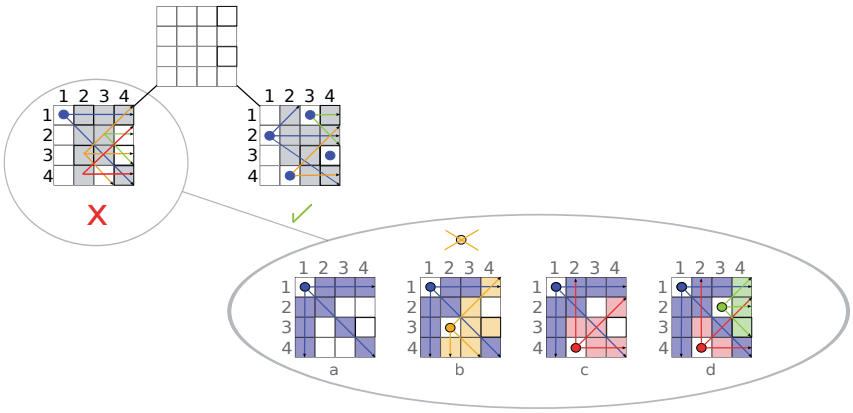

Figura 6: Algoritmo Maintaining Arc Consistency para 4-Queens

4.4. Técnicas de consistencia

Las técnicas de consistencia, también llamadas técnicas de inferencia, se utilizan para mejorar la eficiencia de los algoritmos de búsqueda. Éstas se pueden utilizar antes o durante el proceso de búsqueda, reduciendo la cantidad de nodos a instanciar en el árbol de búsqueda, eliminado los dominios que debido a una restricción no corresponderán a una solución [3].

4.4.1. Nodo-Consistencia

La consistencia de nodo asegura que todos los valores del dominio de una variable satisfacen todas las restricciones unarias sobre esa variable, es decir, se eliminan los valores inconsistentes con las restricciones unarias donde la variable participe [3].

Definición formal: Una variable (X_i) es nodo-consistente si y sólo si, los valores que están en su dominio satisfacen la restricción unaria donde participa la variable:

$$\forall\ X_i \in X, \forall\ R_i \in R, \exists\ a \in D_i : a\ satisface\ R_i \tag{1}$$

Definición formal: Un problema de satisfacción de restricciones en nodo-consistente si todas sus variables son nodo consistentes:

$$\forall\ X_i \in X, \forall\ R_i \in R : X_1\ satisface\ R_i \tag{2}$$

Ejemplo: Un problema cuya variable es X_1, dominio es el conjunto $\{-1, 0, 1, 2, 3, 4\}$ y la restricción unaria es $R : X_1 \geq 2$.

Al aplicar nodo consistencia se eliminarán del dominio los valores que no satisfacen la restricción unaria, por lo tanto el dominio quedaría dado por el conjunto $\{2, 3, 4\}$, llamado dominio nodo-consistente.

4.4.2. Arco-Consistencia

La técnica es muy efectiva y se aplica a las restricciones que involucran dos variables. El arco-consistencia asegura que cada valor en el dominio de cada variable está soportado por un valor de la otra variable que participa en la restricción [3].

Definición formal: Un valor $a \in D$ es arco-consistente relativo a X_i si y sólo si, existe un valor $b \in D$ tal que (X_i, a) y (X_i, b) satisfacen la restricción R_{ij}.

Definición formal: Una variable X_i es arco-consistente relativa a X_j si y sólo si, todos los valores de D_i son arco-consistentes relativos a X_j. Es decir, una variable es arco-consistente si es consistente relativa a todas aquellas con las que interviene en alguna restricción.

Por lo tanto, un problema de satisfacción de restricciones es arco-consistente si todas las variables son arco-consistente.

Ejemplo: Dada una restricción R_{ij}: $x_i < xj$ de la Figura 7 se puede ver que el arco R_{ij} es consistente, ya que para cada valor de $a \in [3, 6]$ hay al menos un valor en $b \in [8, 10]$ de manera que satisface la restricción R_{ij}.

4.5. Estrategias de enumeración

Una asignación de variables, también llamado instanciación, (x, a), es un par variable-valor que representa la asignación del valor a, a la variable x. Una instancia de un conjunto de variables es una tupla de pares ordenados, donde cada par ordenado (x, a) asigna el valor a, a la variable x. Una tupla es localmente consistente si satisface todas las restricciones formadas por variables de la tupla $((x_1, a_1),\ldots ,(x_i, a_i))$. Para simplificar la notación, sustituiremos la tupla $((x_1, a_1), ..., (x_i, a_i))$ por $(a_1, ..., a_i)$ [3, 11, 12].

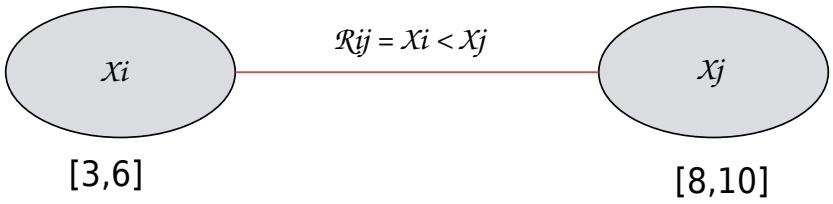

Figura 7: Ejemplo Arco-Consistencia

4.5.1. Heurísticas

Para el funcionamiento de los algoritmos de búsqueda es necesario contar con un orden específico, en el cual cada una de las variables serán estudiadas, el orden en que los valores del dominio serán instanciados para cada variable y, por último, se puede especificar el orden de las restricciones presentes en el problema.

La correcta selección del orden para la variable y el valor, puede mejorar la eficiencia del proceso de búsqueda de la solución. Para establecer estos órdenes es que existen las heurísticas de selección de variable y de valor. Las distintas combinaciones de estas heurísticas forman las denominadas *estrategias de enumeración*. Cada una de las distintas estrategias de enumeración tendrá un comportamiento distinto, por lo tanto pueden existir algunas con un comportamiento más eficiente que otras. A continuación se describirán algunas heurísticas de selección de variable y valor [3, 12].

4.5.2. Heurística de selección de variable

El orden en que las variables son asignadas durante la búsqueda puede tener un efecto significativo en cuanto al espacio de búsqueda. Generalmente estas heurísticas tratan de seleccionar antes las variables que más restringen a las demás, con el fin de reducir el número de fallos (retrocesos). Las heurísticas de selección de variables pueden ser estáticas o dinámicas [3, 12].

- **Selección estática de variable**: Asigna un orden fijo de las variables antes del inicio de la búsqueda. Cada nivel de árbol de búsqueda se asocia con una variable y es necesario que cada nodo se asocie con una variable para la generación de sus sucesores.
- **Selección dinámica de variable**: Puede cambiar el orden de las variables dinámicamente basándose en información local que se genera durante el proceso de búsqueda. En el árbol de búsqueda se ve representada porque existen diferentes variables en un mismo nivel.

Por lo general, estas heurísticas seleccionan antes las variables que restringen de mayor manera el espacio de búsqueda, para así obtener soluciones infactibles de forma temprana.

Las heurísticas de selección de variable que se utilizarán en este nuevo enfoque son:

- **input_order:** Se selecciona la primera variable de la lista.
- **first_fail:** Se selecciona la variable con el dominio más pequeño.
- **anti_first_fail:** Se selecciona la variable con el dominio más grande.
- **smallest:** Se selecciona la variable con el valor más pequeño del dominio.
- **largest:** Se selecciona la variable con el valor más grande del dominio.
- **occurrence:** Se selecciona la variable con la mayor cantidad de restricciones.
- **most_constrained:** Se selecciona la variable con el dominio más pequeño. Si existen varias variables con el mismo tamaño, se selecciona aquella que posea el mayor número de restricciones.
- **max_regret:** Se selecciona la variable con la mayor diferencia entre el valor más pequeño y el segundo más pequeño del dominio. Este método se utiliza si normalmente la variable representa un costo y estamos interesados en la elección que podría incrementar los costos, si no, se escoge la mejor alternativa. Desafortunadamente no siempre funciona; si dos variables de desición incurren en el mismo costo mínimo, el peso no se calcula como cero, si no como la diferencia de este valor mínimo al siguiente valor mayor.

4.5.3. Heurística de selección de valor

Existe poco trabajo asociado a las heurísticas de selección de valor a diferencia de las heurísticas de selección de variable. La idea para estas heurísticas es que seleccionen un valor para la variable actual, que sea más probable que lleve a una solución. En otras palabras, identificar qué rama del árbol de búsqueda nos puede llevar a la solución. En su mayoría estas heurísticas tratan de instanciar un valor que esté poco restringido en relación a la variable actual, es decir, aquel valor que reduce mínimamente el número de valores útiles para las futuras variables [3, 12].

- **indomain**: Los valores se seleccionan en orden creciente. En caso de error, el valor probado anteriormente no se elimina.
- **indomain_middle**: Los valores se seleccionan comenzando desde el centro del dominio (posición). En caso de error, se elimina el valor previamente probado.

- **indomain_max**: Los valores se seleccionan en orden decreciente. En caso de error, se elimina el valor previamente probado.

Según las diferentes combinaciones entre heurísticas de variable y valor, se generarán las estrategias de enumeración. Éstas pueden tener rendimientos significativamente diferentes, por lo tanto es de mucha importancia seleccionar una buena estrategia de enumeración.

5. Búsqueda Autónoma

La Búsqueda Autónoma (en inglés Autonomous Search, AS) es un caso particular de sistemas adaptativos. Tiene como objetivo mejorar el rendimiento de la solución de un problema de satisfacción de restricciones, adaptándose al problema en cuestión. Posee la habilidad de modificar sus componentes internos cuando es expuesto a cambios externos y a oportunidades [31].

Su objetivo es mejorar el rendimiento de la solución, adaptando la estrategia de búsqueda del problema. Esto se puede medir a través del valor de los indicadores o del tiempo en el cual se obtiene la solución del problema. Los componentes internos, que se pueden modificar, corresponden a varios algoritmos involucrados en el proceso de búsqueda: heurísticas, inferencias, etc. Los componentes externos corresponden a la evolución de la información recogida durante el proceso de búsqueda. Esta información también puede ser recogida directamente del problema (por ejemplo: tamaño del espacio de búsqueda exacto o estimado, número de sub-problemas) o indirectamente a través de la eficiencia percibida de los componentes individuales, por ejemplo la capacidad de poda de las técnicas de inferencia. La información también puede referirse al ambiente computacional.

Autonomous Search, permite a los sistemas mejorar su rendimiento, ya que guía la toma de decisiones en la fase de enumeración donde el orden de las variables y los valores son seleccionados, además posee una arquitectura donde sus componentes interactúan entre sí entregándose información, lo que sirve para el análisis de la siguiente estrategia a utilizar.

5.1. Arquitectura

Decidir qué heurísticas de variable y valor utilizar es un trabajo difícil y aún cuando éstas sean elegidas correctamente, no se asegura encontrar una solución para un problema de satisfacción de restricciones, o que si se encuentra la solución ésta sea la óptima. En consecuencia es necesario llevar a cabo un proceso de resolución, donde será indispensable evaluar las estrategias, para encontrar la solución [11, 12].

La arquitectura de AS está basada en 4 componentes (ver Figura 8): solver, observación, análisis y actualización los cuales son descritos particularmente a continuación.

- **Solver**: Componente que corre un algoritmo CSP genérico, alternando la propagación de restricciones y las fases de enumeración. Posee un conjunto de estrategias de enumeración básicas cada una caracterizada por una prioridad que evoluciona durante el cálculo, el componente actualización evalúa las estrategias y actualiza sus prioridades. Para cada enumeración (asignación de un valor a una varia-

ble), la estrategia de enumeración dinámica selecciona la estrategia básica para ser usada en base a las prioridades fijadas.

- **Observación**: Este componente tiene como objetivo registrar alguna información acerca del actual árbol de búsqueda, por ejemplo observar el proceso de resolución en el componente solver. Estas observaciones, llamadas *snapshots*, no son realizadas continuamente y consisten en extraer y registrar (desde el árbol de búsqueda) cierta información del estado de la solución.

- **Análisis**: Este componente es el encargado de estudiar los snapshots tomados por el componente información. También evalúa las diferentes estrategias y provee de indicadores para el componente actualización. Los indicadores pueden ser extraídos, calculados o deducidos desde uno o varios snapshots de la base de datos de snapshots.

- **Actualización**: Componente que toma las decisiones usando una función de selección o choice function. Esta choice function determina el desempeño (o rendimiento) de una estrategia dada en un determinado lapso de tiempo y puede ser calculada basada en los indicadores entregados por el componente análisis (tablas 11 y 12).

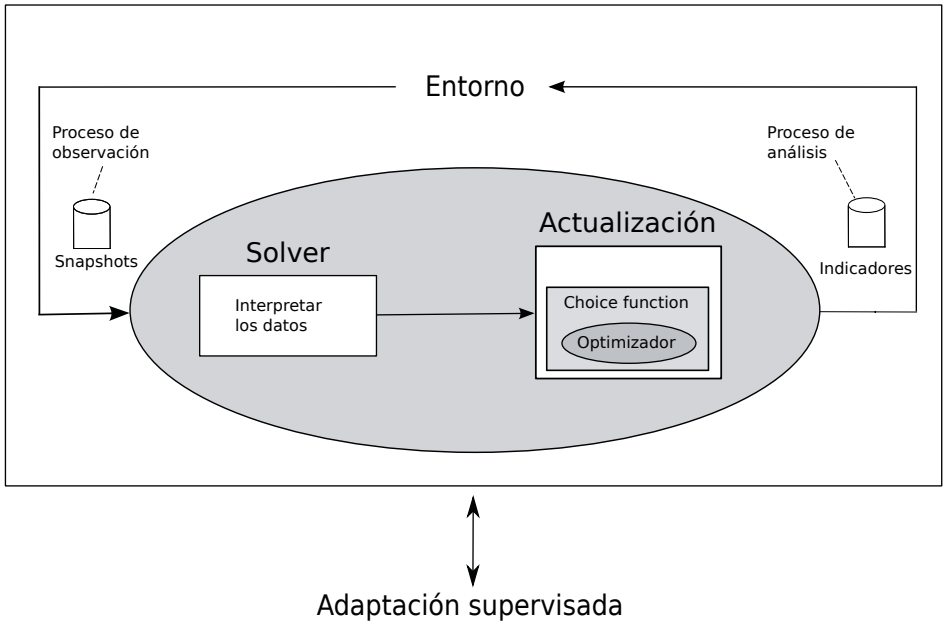

Figura 8: Diagrama del Framework actual

5.2. Choice Function

La *Choice Function* o función de selección tiene como objetivo hacer un ranking con las múltiples estrategias de enumeración, dependiendo del rendimiento, según indicadores, que cada una de éstas poseen [11, 12, 33, 34, 35].

En cada llamado, la choice function rankea, califica y luego elige entre las estrategias de enumeración. La definición de la choice function f para cada estrategia S_j, en el paso n está dada de la siguiente forma:

$$f_n(S_j) = \sum_{i=1}^{I} \alpha_i f_{in}(S_j), \tag{3}$$

donde I es la cantidad de indicadores utilizados y α es el parámetro utilizado para controlar la relevancia del indicador en la choice function, calculado con el optimizador Bat. La lista de indicadores base y calculados, se encuentran en los anexo A.1.1 y A.1.2, en las tablas 11 y 12, respectivamente.

A esto, se agrega que para controlar la relevancia del indicador i, en la estrategia S_j, en un intervalo de tiempo definido, se utiliza una técnica estadística que permite producir una serie de tiempo suavizado, denominada **Exponential Smoothing** o suavizado exponencial.

La idea principal de suavizado exponencial es asignarle una mayor importancia a rendimientos más recientes, asociándoles un factor de "peso" (**smooting factor** o factor de suavizado) exponencialmente decreciente para mayores observaciones, es decir, las observaciones más recientes entregan mayor peso a los rendimientos que las más antiguas. Esta técnica es aplicada al cálculo de $f_{in}(S_j)$, el cual es definido de la siguiente manera:

$$f_{in}(S_j) = v_0 \tag{4}$$

$$f_{in}(S_j) = v_n + \beta_i f_{in-1}(S_j), \tag{5}$$

donde, v_0 es el valor del indicador i, para la estrategia S_j, en el paso n. β es el factor de suavizado y $0 < \beta < 1$.

Se debe considerar que la velocidad a la cual la observación antigua es suavizada, depende de β. Cuando β es cercano a 0, el suavizado es rápido y cuando es cercano a 1, el suavizado es lento.

La Figura 9 muestra el comportamiento general de la interacción entre las estrategias de enumeración y la choice function. Ésta intenta capturar la correspondencia entre el comportamiento histórico de cada estrategia y el punto de decisión actualmente investigado.

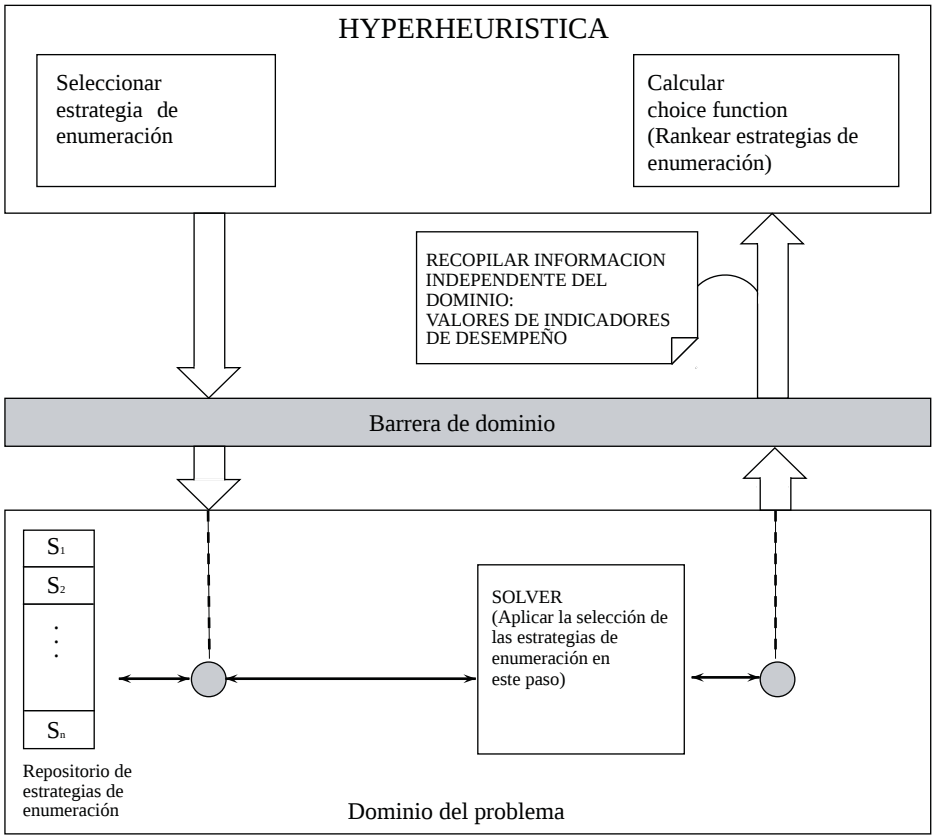

Figura 9: Ranking de las estrategias de enumeración

El esquema de la Figura 8 muestra que la N-ésima estrategia será analizada por la choice function que tras el cálculo, deja a la estrategia N con el mayor valor (o mayor prioridad).

5.2.1. Cálculo parámetro α

La relevancia de cada indicador en la choice function, está dada por el parámetro α, utilizado en la ecuación (3). El cálculo de este parámetro puede ser planteado como un problema de optimización, pues al maximizar o minimizar α, el indicador tomará mayor o menor relevancia. Por ejemplo, si se considera el indicador **B** (ver tabla indicadores base, A.1.1), un α_i cercano a 1, muestra que ese indicador es sumamente relevante, por lo que buenos resultados (pocos backtrack) son fundamentales para la choice function.

Para resolver este problema, se ha planteado un nuevo enfoque de control en línea de las estrategias de enumeración basado en el optimizador Bat. El algoritmo Bat es una metaheurística relativamente nueva, fundamentada en el comportamiento de ecolocalización que poseen ciertos murciélagos que emplean sonidos para identificar los objetos en su entorno. En el siguiente capítulo, se describe en profundidad el funcionamiento de esta metaheurística.

6. Algoritmo Bat

Los murciélagos son animales asombrosos y su capacidad avanzada de ecolocalización han atraído la atención de investigadores de diferentes áreas de estudio. El algoritmo Bat estándar se basa en las características de ecolocalización o bio-sonar que poseen los micro murciélagos [36, 37]. Éstos emiten un sonido fuerte y corto, a la espera que llegue en un objeto y luego de una fracción de tiempo, el eco regresa de nuevo a sus oídos. Por lo tanto, los murciélagos pueden calcular la distancia a la que se encuentra un objeto. Además, este increíble mecanismo de orientación hace que los murciélagos sean capaces de distinguir la diferencia entre un obstáculo y una presa, lo que les permite cazar incluso en completa oscuridad.

En base a la descripción y características de murciélagos ecolocación anterior, *Xin-She Yang* (2010) desarrolló el algoritmo Bat con las siguientes tres reglas [36, 37, 38]:

i. Aún cuando se sabe que sólo los micro murciélagos tienen la capacidad de ecolocalización, se generaliza a que todos los murciélagos utilizan la ecolocalización para detectar la distancia, y también "saben" la diferencia entre los alimentos, presas y obstáculos, de alguna manera mágica;

ii. Los murciélagos vuelan de forma aleatoria con velocidad v_i en la posición x_i con una frecuencia f_{min}, variando su longitud de onda λ y volumen A_0, para buscar presas. Ellos pueden ajustar automáticamente la longitud de onda (o frecuencia) de sus impulsos emitidos y ajustar la tasa de emisión de impulsos $r \in [0, 1]$, dependiendo de la proximidad de su objetivo;

iii. Aunque el volumen puede variar de muchas maneras, se asume que el volumen varía de A_0 (positivo) a un valor constante mínima A_{min}.

En primer lugar, la posición inicial x_i, la velocidad v_i y la frecuencia f_i se inicializa para cada murciélago b_i, en cada paso de tiempo t, siendo T el número máximo de iteraciones. El movimiento de los murciélagos se da mediante la actualización de su velocidad y por consecuencia su posición, usando las ecuaciones (6), (7) y (8), de la siguiente manera:

$$f_i = f_{min} + (f_{max} - f_{min})\beta \tag{6}$$

$$v_i^j(t) = v_i^j(t-1) + [(\hat{x})^j - x_i^j(t-1)]f_i \tag{7}$$

$$x_i^j(t) = x_i^j(t-1) + v_i^j(t) \tag{8}$$

donde β es un número generado aleatoriamente, extraído de una distribución uniforme en el intervalo $[0, 1]$. El resultado de f_i (ecuación (6)) se utiliza para controlar el rango del movimiento de los murciélagos.

Recordemos que $v_i^j(t)$ denota el cambio de velocidad en la variable de decisión j para b_i en el instante t. La variable $\hat{x}^j$ representa la actual mejor posición (solución) para la componente o variable de decisión j, que se consigue comparando todas las soluciones proporcionadas por los murciélagos m. Por último, la variable $x_i^j(t)$ indica el nuevo valor de la variable de decisión j para b_i en instante t.

El algoritmo planteado por *Xin-She Yang*, es el siguiente [36]:

Algoritmo 1 Bat algorithm

1: Objetive function $f(x), x = x^1, ..., x^n$.
2: Initialize the bat population x_i and velocity v_i, $i = 1, 2, ..., m$.
3: Define pulse frecuency f_i, at x_i, $\forall i = 1, 2, ..., m$.
4: Initialize pulse rates r_i and the loudness A_i, $i = 1, 2, ..., m$.
5: **while** $t < T$ **do**
6: **for all** x_i **do**
7: Generate new solutions through equations (6), (7) and (8).
8: **if** $rand > r_i$ **then**
9: Select a solution among the best solutions.
10: Generate a local solution around the best solution through equation (9).
11: **end if**
12: **if** $rand < A_i$ and $f(x_i) < f(\hat{x})$ **then**
13: Accept the new solutions.
14: Increase r_i and reduce A_i through equations (11), (10).
15: **end if**
16: **end for**
17: **end while**
18: **return** Rank the bats and find the current best $\hat{x}$.

Con el fin de mejorar la variabilidad de las posibles soluciones, el autor ha propuesto el uso de "paseos aleatorios" (*random walk*) [36, 37].

En primer lugar, se selecciona una solución de entre las mejores soluciones actuales (línea 9 del algoritmo) y luego el se aplica una aleatoriedad con el fin de generar una nueva solución para cada x_i que acepta la condición en la línea 8 de algoritmo 1, aplicando el siguiente método:

$$x_{new}^j = x_{old}^j + \epsilon \bar{A}(t), \tag{9}$$

en la que $\bar{A}(t)$ indica el volumen promedio de todos los murciélagos en el instante t, $\epsilon \in [-1, 1]$. Para cada iteración del algoritmo Bat, el volumen A_i y el radio r_i se actualizan, de la siguiente manera:

$$A_i(t+1) = \alpha A_i(t) \tag{10}$$

$$r_i(t+1) = r_i(0)[1 - exp(-\gamma t)], \tag{11}$$

donde $0 < \alpha < 1$ y $\gamma > 0$, son constantes ad-hoc.

En los pasos 3 y 4 del algoritmo, el $r_i(0)$ y $A_i(0)$ se eligen a de manera aleatoria, generalmente, $A_i(0) \in [1, 2]$ y $r_i(0) \in [0, 1]$.

7. Experimentos

Se ha llevado a cabo una evaluación experimental del enfoque propuesto, en distintas instancias de problemas de satisfacción de restricciones A.2:

- N-Queens (n-Q) con n ∈ {8, 10, 12, 15, 20, 50, 75}.
- Magic Square (n-MS) con n ∈ {3, 4, 5, 6, 7}.
- Sudoku, problemas {1, 2, 5, 7, 9}.
- Knight's Tour (n-KT) con n ∈ {5, 6}.
- Latin Square (n-LS) con n ∈ {4, 5, 6, 7, 8}
- Quasigroup (n-QS) con n ∈ {1, 3, 5, 6, 7}
- Langford, problemas {{2, 8}, {2, 12}, {2, 16}, {2, 20}, {2, 23}}

Los experimentos se realizaron en un equipo bajo las siguientes características:

- Sin conexión a internet.
- Sin otros programas ejecutándose.
- Se realizan 30 pruebas por problema y se escoge la mejor.

El componente de la enumeración adaptativa se ha implementado en *ECL^{i}PS^e Constraint logic Programming Solver v6.10*, y el optimizador Bat ha sido desarrollado en el lenguaje de programación Java. Los experimentos se han implementado en un equipo con un procesador *Intel Core i3-2120 de 3.30 GHz*, con *4Gb* de RAM corriendo sobre *Windows 7 Professional de 32 bits*. Las instancias se resolvieron a un número máximo de 65535 pasos. Si se llega al límite de pasos y no se ha encontrado solución, se considera como fallido y se asume un *time-out (t.o.)* como tiempo de resolución.

Antes de iniciar los experimentos, se realizó un muestreo con las variables de entradas del algoritmo Bat, para encontrar la mejor configuración. La tabla 1, muestra el mejor resultado obtenido con las instancias *8-Q*, *16-Q* y *20-Q*, del problema *n-Queens*.

Las pruebas se realizaron sobre un portafolio de veinticuatro estrategias de enumeración y cuatro funciones de selección (choice functions), que se muestran en la tablas 2 y 3, respectivamente.

Configuración							Resultados		
Iteracions (T)	**Población (n)**	f_{min}	f_{max}	α	γ	ϵ	**BT**	**Step**	***Runtime (ms)***
75	10	0,75	1,25	0,9	0,9	0,9	**51,9**	**115,4**	**636**
75	20	0,35	0,65	0,9	0,9	0,9	**510**	**936,3**	**727**
75	20	1,25	1,75	0,9	0,9	0,9	**786,8**	**1557,3**	**796**
75	10	0,35	0,65	0,9	0,9	0,9	**795,6**	**1451,9**	**804**
75	20	0,75	1,25	0,9	0,9	0,9	**705,8**	**1300,2**	**814**
75	30	0,35	0,65	0,9	0,9	0,9	**59,1**	**126,8**	**856**
100	10	0,75	1,25	0,9	0,9	0,9	**11**	**39**	**887**
75	30	0,75	1,25	0,9	0,9	0,9	**695,4**	**1277**	**892**
75	30	1,25	1,75	0,9	0,9	0,9	**718,9**	**1324,6**	**918**
100	10	0,35	0,65	0,9	0,9	0,9	**363**	**679,5**	**972**
75	40	0,35	0,65	0,9	0,9	0,9	**724,6**	**1228,9**	**1074**
150	10	0,35	0,65	0,9	0,9	0,9	**355,8**	**650,7**	**1108**
150	30	0,75	1,25	0,9	0,9	0,9	**765,7**	**1430,7**	**1476**
150	40	0,35	0,65	0,9	0,9	0,9	**357**	**667**	**1527**
100	20	0,35	0,65	0,9	0,9	0,9	**642,8**	**1292,7**	**1605**
100	20	0,75	1,25	0,9	0,9	0,9	**430,6**	**812,7**	**1714**
100	30	1,25	1,75	0,9	0,9	0,9	**494,8**	**921,4**	**1821**
100	20	1,25	1,75	0,9	0,9	0,9	**259,6**	**503**	**1871**
150	10	1,25	1,75	0,9	0,9	0,9	**499,3**	**924,5**	**1907**
100	40	0,75	1,25	0,9	0,9	0,9	**516,5**	**939,8**	**1970**
150	30	1,25	1,75	0,9	0,9	0,9	**176,5**	**336,7**	**1982**
150	20	0,35	0,65	0,9	0,9	0,9	**177,7**	**320,7**	**1995**
100	30	0,35	0,65	0,9	0,9	0,9	**294,5**	**556**	**1998**
100	40	1,25	1,75	0,9	0,9	0,9	**225,5**	**428,3**	**2012**
150	40	0,75	1,25	0,9	0,9	0,9	**334,6**	**611,8**	**2155**
75	40	0,75	1,25	0,9	0,9	0,9	**67**	**140,8**	**2137**
150	40	1,25	1,75	0,9	0,9	0,9	**536,3**	**987,9**	**2237**
200	30	0,35	0,65	0,9	0,9	0,9	**673,3**	**1248**	**2347**
100	30	0,75	1,25	0,9	0,9	0,9	**64,1**	**136,3**	**2455**
200	30	0,75	1,25	0,9	0,9	0,9	**628,6**	**1159,1**	**2465**
75	40	1,25	1,75	0,9	0,9	0,9	**99,5**	**201,2**	**2521**
200	10	1,25	1,75	0,9	0,9	0,9	**705,7**	**1330,9**	**2877**
200	10	0,75	1,25	0,9	0,9	0,9	**141,9**	**291,7**	**2891**
200	20	1,25	1,75	0,9	0,9	0,9	**198**	**378**	**3072**
200	20	0,35	0,65	0,9	0,9	0,9	**413,3**	**761**	**3010**
150	20	0,75	1,25	0,9	0,9	0,9	**593,4**	**1076,5**	**3112**
200	40	0,35	0,65	0,9	0,9	0,9	**197**	**372,9**	**3117**
200	20	0,75	1,25	0,9	0,9	0,9	**132,2**	**264,5**	**3217**
200	30	1,25	1,75	0,9	0,9	0,9	**231,2**	**444,2**	**3297**
200	40	0,75	1,25	0,9	0,9	0,9	**253,4**	**471,9**	**3485**
200	40	1,25	1,75	0,9	0,9	0,9	**237,7**	**440,7**	**3572**

Tabla 1: Configuración Bat algorithm

Heurísticas de selección								
Id	**Variable**	**Valor**	Id	**Variable**	**Valor**	Id	**Variable**	**Valor**
S_1	**input_order**	**Mín**	S_9	**input_order**	**Med**	S_{17}	**input_order**	**Máx**
S_2	**first_fail**		S_{10}	**first_fail**		S_{18}	**first_fail**	
S_3	**anti_first_fail**		S_{11}	**anti_first_fail**		S_{19}	**anti_first_fail**	
S_4	**occurrence**		S_{12}	**occurrence**		S_{20}	**occurrence**	
S_5	**smallest**		S_{13}	**smallest**		S_{21}	**smallest**	
S_6	**largest**		S_{14}	**largest**		S_{22}	**largest**	
S_7	**most_constrained**		S_{15}	**most_constrained**		S_{23}	**most_constrained**	
S_8	**max_regret**		S_{16}	**max_regret**		S_{24}	**max_regret**	

Tabla 2: Estrategias de enumeración

Choice functions										
CF_1:	$\alpha_1 B$	+	$\alpha_2 Step$	+	$\alpha_3 PTAVFES$					
CF_2:	$\alpha_1 SB$	+	$\alpha_2 In1$	+	$\alpha_3 In2$					
CF_3:	$\alpha_1 B$	+	$\alpha_2 Step$	+	$\alpha_3 In1$	+	$\alpha_4 In2$			
CF_4:	$\alpha_1 VF$	+	$\alpha_2 dmax$	+	$\alpha_3 DB$					

Tabla 3: Choice functions utilizadas en los experimentos con el enfoque BAT.

Se realizó una comparación entre el enfoque **BAT** propuesto, basado en el optimizador Bat, con dos sistemas desarrollados anteriormente, uno basado en el algoritmo genético (**GA**) y el otro basado en la optimización de enjambre de partículas (**PSO**). Se consideró además, para la comparación de los resultados obtenidos, la realización de pruebas sin un control en línea, vale decir, una única estrategia de enumeración (S_1 a S_{24}), para la resolución completa del CSP. Al final, se incluyó una estrategia de selección aleatoria y un conjunto de choice function sin el cálculo del parámetro α_i ($\alpha_i = 1$). El listado de este conjunto se describe en la Tabla 4.

Choice functions										
CF_a:	VFP	+	SSR	-	B					
CF_b:	VFP	+	SSR	-	SB					
CF_c:	VFP	+	SSR	-	SB	-	B			
CF_d:	VFE	+	SSR	-	B					
CF_e:	VFE	+	SSR	-	SB					
CF_f:	VFE	+	SSR	-	SB	-	B			
CF_g:	VFP	+	VFE	-	SSR	-	B			
CF_h:	VFP	+	VFE	-	SSR	-	SB			
CF_i:	VFP	+	VFE	-	SSR	-	SB	-	B	
CF_j:	VFP	+	SSR	-	$TRASH$					
CF_k:	VFE	+	SSR	-	$TRASH$					
CF_l:	VFP	+	VFE	-	SSR	-	$TRASH$			

Tabla 4: Choice functions utilizadas en los experimentos sin parámetro α.

Para la evaluación de las pruebas, se consideró el número total de **backtracks** y **runtime** (tiempo de resolución en milisegundos) necesarios para llegar a una solución.

Para las instancias N-Queens (ver Tabla 5), en general las de mayor complejidad ($n > 20$), las estrategias de enumeración por sí solas, no logran resolverlas dentro de un contexto máximo de backtrack ($cutoff_1 = 65{,}535$), quedando demostrado que, si bien es posible que cada una de ellas pueda encontrar una solución, en algunos casos ésta es inviable en este contexto. Continuando con la evaluación de las instancias N-Queens de mayor complejidad ($n > 20$), bajo el mismo contexto del máximo número de backtrack

($cutoff_1$), los resultados demuestran que el 75 % de las choice functions no logran resolverlas.

De los resultados obtenidos para el problema Sudoku Puzzle (ver Tabla 5), se puede apreciar que para las instancias de mayor complejidad ($n \in \{5, 7\}$), la mitad de las estrategias de enumeración, por sí solas, no son capaces de encontrar una solución, bajo el contexto de una cantidad máxima de backtrack inferior al $cutoff_1$, al igual que las choice functions.

S_j	**Problemas**													
	N-Queens							$\bar{X}_Q$	**Sudoku**					$\bar{X}_S$
	8Q	**10Q**	**12Q**	**15Q**	**20Q**	**50Q**	**75Q**		**1S**	**2S**	**5S**	**7S**	**9S**	
S_1	10	6	15	73	10026	>121277	>118127	>121277	0	18	4229	10786	0	3006,6
S_2	11	12	11	808	2539	>160845	>152812	>160845	1523	10439	>89125	>59828	114	>89125
S_3	10	4	16	1	11	177	818	148,14	0	4	871	773	0	329,6
S_4	10	6	15	73	10026	>121277	>118127	>121277	0	18	4229	10786	0	3006,6
S_5	3	6	17	40	862	>173869	>186617	>186617	7	155	>112170	>81994	0	>112170
S_6	9	6	16	82	15808	>143472	>137450	>143472	0	764	>83735	>80786	0	>83735
S_7	10	4	16	1	11	177	818	148,14	0	4	871	773	0	329,6
S_8	3	5	12	28	63	>117616	>133184	>133184	0	2	308	10379	0	2137,8
S_9	10	6	15	73	10026	>121277	>118127	>121277	0	18	4229	10786	0	3006,6
S_{10}	11	12	11	808	2539	>160845	>152812	>160845	1523	10439	>89125	>59828	114	>89125
S_{11}	10	12	16	1	11	177	818	149,29	0	4	871	773	0	329,6
S_{12}	10	6	15	73	10026	>121277	>118127	>121277	0	18	4229	10786	0	3006,6
S_{13}	3	6	17	40	862	>173869	>186617	>186617	7	155	>112174	>81994	0	>112174
S_{14}	9	6	16	82	15808	>143472	>137450	>143472	0	764	>83735	>80786	0	>83735
S_{15}	10	4	16	1	11	177	818	148,14	0	4	871	773	0	329,6
S_{16}	3	5	12	28	63	>117616	>133184	>133184	0	2	308	10379	0	2137,8
S_{17}	10	6	15	73	10026	>121277	>118127	>121277	3	2	>104148	1865	1	>104148
S_{18}	11	12	11	808	2539	>160845	>152812	>160845	8482	6541	>80203	>80295	7	>80295
S_{19}	10	4	16	1	11	177	818	148,14	0	9	963	187	0	231,8
S_{20}	10	6	15	73	10026	>121277	>118127	>121277	3	2	>104148	1865	1	>104148
S_{21}	9	6	16	82	15808	>173869	>186617	>186617	49	89	>78774	>93675	0	>93675
S_{22}	3	6	17	40	862	>143472	>137450	>143472	1039	887	>101058	>91514	0	>101058
S_{23}	10	4	16	1	11	177	818	148,14	4	9	963	187	0	232,6
S_{24}	2	37	13	127	1129	>117616	>133184	>133184	72	12	>92557	2626	0	>92557
CF_a	6	7	14	64	1515	>100617	>108851	>108851	2	35	46620	>379622	0	>379622
CF_b	11	15	21	111	23	>117715	>115793	>117715	3	216	>282481	953	0	>282481
CF_c	4	21	15	99	14869	>118985	>119766	>119766	20	26	183	>172858	1	>172858
CF_d	3	6	13	47	1482	237	13784	2224,57	0	248	6305	62719	0	13854,4
CF_e	3	9	15	104	1207	299	>115861	>115861	4	22	>120157	4596	0	>120157
CF_f	6	10	17	77	11019	>123967	>149734	>149734	4	4	>115878	>192593	0	>192593
CF_g	3	6	14	47	2496	>103648	>129444	>129444	5	82	>178887	>190212	0	>190212
CF_h	7	8	20	70	81	>115509	>133297	>133297	0	27	2234	>158181	7	>158181
CF_i	10	6	21	51	1009	>106592	>112009	>112009	3	183	3706	>141212	4	>141212
CF_j	9	7	15	76	193	202	3036	505,43	0	189	626	15892	0	3341,4
CF_k	3	6	17	85	3469	>139215	>152046	>152046	2	18	>495296	>222342	6	>495296
CF_l	3	6	14	2	11945	>187792	>148987	>187792	0	4	>259935	>315924	4	>315924
RND	1	1	2	12	39	0	0	7,86	0	1	7	227	0	47
PSO	3	1	1	1	11	0	818	119,29	0	2	13	256	0	54,2
GA	1	4	40	68	38	15	17	26,14	732	6541	4229	10786	7	4459
BAT	**2**	**4**	**0**	**1**	**8**	**0**	**31**	**6,57**	**0**	**1**	**80**	**117**	**0**	**39,6**

Tabla 5: Backtracks requeridos para las diferentes estrategias de enumeración en los problemas N-Queens y Sudoku.

Los enfoques BAT, PSO y GA superan el número de backtracks de las estrategias de enumeración y en algunos casos a las choice functions, en las instancias de menor complejidad, siendo BAT el que menos backtrack provoca de los 3 optimizadores. Sin embargo, para los casos más complejos, como 50-Q y

75-Q, el nuevo control en línea funciona mejor que las estrategias de enumeración y supera notablemente a las choice functions. PSO es superado por GA y BAT, y BAT supera en la mayoría de las pruebas a GA.

Considerando las instancias del problema Sudoku Puzzle (ver Tabla 5), BAT supera por amplio margen a las estrategias de enumeración y las choice function y en promedio está mejor evaluada que PSO. Ambos enfoques aventajan a GA.

S_j	**Problemas**											
	Magic Square					$\bar{X}_{MS}$	**Latin Square**					$\bar{X}_{LS}$
	3MS	**4MS**	**5MS**	**6MS**	**7MS**		**4LS**	**5LS**	**6LS**	**7LS**	**8LS**	
S_1	0	12	910	>177021	>170762	>177021	0	0	0	12	0	2,4
S_2	4	1191	>191240	>247013	>39181	>247013	0	9	163	>99332	0	>99332
S_3	0	3	185	>173930	>169053	>173930	0	0	0	0	0	0
S_4	0	10	5231	>187630	>127273	>187630	0	0	0	14	0	2,8
S_5	0	22	>153410	>178895	>79041	>178895	0	7	61	>99403	0	>99403
S_6	4	992	>204361	>250986	>230408	>250986	0	0	0	71	0	14,2
S_7	0	3	193	>202927	>166861	>202927	0	0	0	0	0	0
S_8	0	13	854	>190877	>202632	>202632	0	0	0	0	0	0
S_9	0	12	910	>177174	>170762	>177174	0	0	0	9	0	1,8
S_{10}	4	1191	>191240	>247013	>63806	>247013	0	9	163	>99332	0	>99332
S_{11}	0	3	185	>174068	>169053	>174068	0	0	0	0	0	0
S_{12}	0	10	5231	>187777	>153768	>187777	0	0	0	9	0	1,8
S_{13}	0	22	>153410	>179026	>77935	>179026	0	7	61	>99539	0	>99539
S_{14}	4	992	>204361	>251193	>230408	>251193	0	0	0	71	0	14,2
S_{15}	0	3	193	>203089	>96730	>203089	0	0	0	0	0	0
S_{16}	0	13	854	>191042	>247427	>247427	0	0	0	0	0	0
S_{17}	1	51	>204089	>237428	>229035	>237428	0	0	0	9	0	1,8
S_{18}	0	42	>176414	>176535	>116846	>176535	0	9	163	>99481	0	>99481
S_{19}	1	3	>197512	>231600	>185681	>231600	0	0	0	0	0	0
S_{20}	1	29	74063	>190822	>187067	>190822	0	0	0	9	0	1,8
S_{21}	1	95	>201698	>239305	>249686	>249686	0	0	0	71	0	14,2
S_{22}	0	46	74711	>204425	>46233	>204425	0	7	61	>99539	0	>99539
S_{23}	1	96	>190692	>204119	>130196	>204119	0	0	0	0	0	0
S_{24}	1	47	>183580	>214287	>219116	>219116	0	0	0	0	1	0,2
CF_a	1	19	65	>375482	>359727	>359727	0	0	0	0	0	0
CF_b	0	34	>302322	>348874	>401558	>401558	0	0	0	0	0	0
CF_c	1	16	59	>417572	>591665	>591665	0	0	0	0	3	0,6
CF_d	0	5	859	>274389	>475302	>475302	0	0	0	0	0	0
CF_e	0	11	529	>266332	>327705	>327705	0	7	42	0	0	9,8
CF_f	0	6	5852	>132021	>313998	>313998	0	0	33	0	0	6,6
CF_g	1	9	3378	>179368	>425846	>425846	0	0	0	>176493	0	>176493
CF_h	0	23	4499	>185386	>320555	>320555	0	6	0	>138536	1	>138536
CF_i	0	63	>369813	>286342	>327691	>327691	0	0	29	0	0	5,8
CF_j	0	13	>239193	>251401	>343011	>343011	1	28	0	18	0	9,4
CF_k	0	13	>362597	>492522	>325094	>492522	0	0	0	0	0	0
CF_l	1	22	>132531	>278520	>106901	>278520	0	0	0	52433	0	10486,6
RND	0	0	23	>198857	>200145	>200145	0	0	0	0	0	0
PSO	0	0	14	>47209	>56342	>56342	0	0	0	0	0	0
GA	0	42	198	>176518	>213299	>213299	0	0	0	0	0	0
BAT	**0**	**3**	**99**	**733**	**1445**	**456**	**0**	**0**	**0**	**0**	**0**	**0**

Tabla 6: Backtracks requeridos para las diferentes estrategias de enumeración en los problemas Magic Square y Latin Square.

Para las instancias del Magic Square (ver Tabla 6), en general las de mayor complejidad ($n > 4$), al igual que en el caso anterior, las estrategias de enumeración por sí solas y las choice functions no logran resolverlas dentro de un contexto máximo de backtrack ($cutoff_1$), aumentando en gran medida la cantidad

de éstos.

Teniendo en cuenta los resultados para la instancia del problemas Magic Square con $n \in \{4, 5\}$, sólo PSO provoca menos backtrack que BAT. Sin embargo, para $n > 5$, BAT logra superar ampliamente a las estrategias de enumeración, las choice functions, la selección radómica y los enfoques propuestos anteriormente PSO, GA.

Para las instancias del problema Latin Square (ver Tabla 6), las estrategias de enumeración y las choice functions, al contario de los casos anteriores, logran resolverlas en su mayoría (cerca del 95 % de las instancias). Sin embargo los resultados obtenidos con el nuevo enfoque BAT son determinantes. No produce backtrack y logra resolver el 100 % de las instancias. En este caso en particular, los resultados obtenidos empatan con los enfoques anteriores PSO y GA.

El problema Knight's Tour, es uno de los problemas combinatoriales más complejos y encontrar una solución es un proceso que requiere de un arduo trabajo. De acuerdo a los resultados obtenidos (ver Tabla 7), se puede inferir que las estrategias de enumeración por sí solas, no son capaces de encontrar una solución, dentro del contexto donde el máximo de backtrack no supere el valor $cutoff_1$. Diferente es el caso de las choice function, quienes al menos el 50 % de ellas, si logran alcanzar una solución, minimizando la cantidad de backtrack.

Para las instancias de menor complejidad del problema Quasigroup ($n < 5$), las estaregias de enumeración, al igual que las choice functions, logran encontrar una solución de forma eficiente, minimizando la cantidad de backtrack, llegando incluso a no producir 0 (ver Tabla 7) en muchas de ellas. Pero queda demostrado también que para las instancias de mayor complejidad ($n = 5$, por ejemplo), tanto las de estrategias de enumeración, como las choice functions no logran encontrar una solución dentro del entorno de que la cantidad máxima de backtrack no supere el valor de corte ($cutoff_1$).

De las pruebas realizadas a las diferentes instancias del problema Langford (ver Tabla 7), se puede inferir que encontrar una solución con sólo una estrategia de enumeración es totalmente viable. Sin embargo existen instancias del problema (en general $\{2, 20\}$ y $\{2, 23\}$), en las que la estrategia de enumeración por sí sola no logra resolver el problema. Considerando las choice functions para resolver el problema Langford (ver Tabla 7), se demuestra que es viable encontrar una solución a cerca del 90 % de las instancias. Sin embargo, esto también demuestra que existe un 10 % de instancias que no logran ser resueltas, aumentando la cantidad de backtrack provocados.

S_j	Problemas															
	Knight's Tour		$\bar{X}_{KT}$	**Quasigroup**					$\bar{X}_{QG}$	**Langford** $L_{n=2}$					$\bar{X}_L$
	5KT	**6KT**		**1QG**	**3QG**	**5QG**	**6QG**	**7QG**		**8L**	**12L**	**16L**	**20L**	**23L**	
S_1	767	37695	19231	0	0	>145662	30	349	>145662	2	16	39	77	26	32
S_2	>179097	>177103	>179097	0	0	>103603	>176613	3475	>176613	15	223	24310	>158157	>157621	>158157
S_3	767	37695	19231	0	0	8343	0	1	1668,8	2	1	97	172	64	67,2
S_4	>97176	35059	>97176	0	0	>145656	30	349	>145656	2	16	39	77	26	32
S_5	>228316	>239427	>239427	1	0	>92253	>83087	4417	>92253	2	29	599	26314	29805	11349,8
S_6	>178970	>176668	>178970	0	0	>114550	965	4417	>114550	1	22	210	1	3	47,4
S_7	>73253	14988	>73253	0	0	8343	0	1	1668,8	2	1	97	172	64	67,2
S_8	>190116	>194116	>194116	1	0	>93315	>96367	4	>96367	0	12	0	64	7	16,6
S_9	767	37695	19231	1	0	>145835	30	349	>145835	2	16	39	77	26	32
S_{10}	>179126	>177129	>179126	0	0	>103663	>176613	3475	>176613	15	223	24310	>158157	>157621	>158157
S_{11}	767	37695	19231	0	0	8343	0	1	1668,8	2	1	97	172	64	67,2
S_{12}	>97176	35059	>97176	0	0	>145830	30	349	>145830	2	16	39	77	26	32
S_{13}	>228316	>239427	>239427	1	0	>92355	>83087	583	>92355	2	29	599	26314	29805	11349,8
S_{14}	>178970	>176668	>178970	0	0	>114550	965	4417	>114550	1	22	210	1	3	47,4
S_{15}	>73253	14998	>73253	0	0	8343	0	1	1668,8	2	1	97	172	64	67,2
S_{16}	>190116	>194116	>194116	1	0	>93315	>93820	4	>93820	0	12	0	64	7	16,6
S_{17}	767	37695	19231	0	0	7743	2009	3	1951	2	16	39	77	26	32
S_{18}	>179126	>177129	>179126	0	0	>130635	>75475	845	>130635	15	223	24592	>158028	>157649	>158028
S_{19}	767	37695	19231	0	0	0	89	1	18	2	1	98	172	64	67,4
S_{20}	>97178	35059	>97178	0	0	7763	2009	3	1955	2	16	39	77	26	32
S_{21}	>178970	>176668	>178970	0	0	>96083	>108987	773	>108987	1	22	210	1	3	47,4
S_{22}	>228316	>239427	>239427	0	0	>94426	>124523	1	>124523	2	29	599	26314	29805	11349,8
S_{23}	>73253	14998	>73253	0	0	0	89	1	18	2	1	98	172	64	67,4
S_{24}	>190116	>160789	>190116	0	0	>95406	>89888	1	>95406	4	6	239	4521	0	954
CF_a	>263692	>374982	>374982	0	0	>326243	0	4	>326243	1	6	89	41981	631	8541,6
CF_b	>145594	>253846	>253846	0	0	2	29	380	82,2	1	44	154	2328	>183298	>183298
CF_c	>235271	>392729	>392729	0	0	>383192	145	674	>383192	9	9	3310	785	174	857,4
CF_d	>220290	>325756	>325756	0	0	0	30	1193	244,6	0	93	3538	62523	72	13245,2
CF_e	>159706	>453045	>453045	0	0	>357144	>365101	28	>365101	2	80	134	3212	1	685,8
CF_f	>146012	>380828	>380828	1	0	>371803	0	1	>371803	4	28	127	309	2	94
CF_g	>332444	>403609	>403609	0	0	>262824	30	22	>262824	4	29	63	202	>171492	>171492
CF_h	>297516	>516415	>516415	0	0	>246955	>313670	4	>313670	1	6	66	>112393	3	>112393
CF_i	>239885	>441666	>441666	0	0	>320560	371	1	>320560	10	1	179	112	200	100,4
CF_j	5490	>452509	>452509	0	0	>168628	59010	37	>168628	1	34	251	37	2	65
CF_k	>858996	>208701	>208701	0	0	>201266	1653	4	>201266	0	6	74	968	114	232,4
CF_l	>436788	>320046	>320046	0	0	>223003	30	380	>223003	2	1	212	7354	1010	1715,8
RND	25	20968	10496,5	0	0	0	0	0	0	0	0	0	0	0	0
PSO	106	12952	6529	0	0	0	0	0	0	0	1	0	1	3	1
GA	5615	86928	46271,5	0	0	7763	0	4	1554,3	15	223	97	64	0	79,8
BAT	**7**	**1499**	**753**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**1**	**0**	**0,4**

Tabla 7: Backtracks requeridos para las diferentes estrategias de enumeración en los problemas Knight's Tour, Quasigroup y Langford.

Ahora, evaluando el problema Knight's Tour para $n = 5$, BAT y PSO claramente son superiores a GA y para el caso más complejo, BAT ocupa el primer lugar, superando incluso a la selección randómica.

Los resultados obtenidos en las pruebas a las instancias de los problemas Quasigroup y Langford son excluyentes, pues BAT es ampliamente superior a todas las técnicas de resolución.

Por último, teniendo en cuenta el número promedio de backtrack para la resolución de todo el conjunto de problemas, BAT se queda con el primer lugar. Una comparación gráfica de los enfoques de control en línea se ilustra en la Figura 10.

Además de medir el desempeño de la búsqueda de una solución a los problemas combinatoriales, en términos de backtrack, las estrategias de enumeración, las choice functions y los enfoques de control en línea, pueden ser comparadas en función del tiempo de resolución (runtime) que se requiere para lograr obtener una solución.

Tal como en las pruebas anteriores, se definió un valor de corte asociado al tiempo máximo de espera para el proceso de búsqueda de una solución ($cutoff_2 = 300000\ ms$). Si el tiempo de resolución supera este nuevo valor de corte, se considera que la técnica en cuestión no es capaz de resolver el problema, asignado como tiempo de resolución un valor *time out* (t.o.).

Para las instancias N-Queens (ver Tabla 8), en general las de mayor complejidad ($n > 20$), las estrategias de enumeración por sí solas, no logran resolverlas dentro del tiempo esperado, quedando demostrado que, si bien es posible que cada una de ellas pueda encontrar una solución, en algunos casos ésta es inviable dentro de este contexto. Continuando con la evaluación de las instancias N-Queens de mayor complejidad ($n > 20$), bajo el mismo contexto de no superar el valor máximo de corte de tiempo de resolución ($cutoff_2$), los resultados demuestran que el 75 % de las choice functions no logran resolverlas.

El enfoque propuesto BAT resulta bastante competitivo, evaluando los tiempos de resolución. Si bien, para las instancias pequeñas de N-Queen, BAT es más lento que GA y PSO, a partir de la instancia media ($n = 20$) y los casos complejos ($n \geqslant 50$) BAT comienza a exhibir un excelente rendimiento.

Los resultados obtenidos para el problema Sudoku Puzzle (ver Tabla 8), permiten apreciar que para las instancias de mayor complejidad ($n \in \{5, 7\}$), la mitad de las estrategias de enumeración, por sí solas, no son capaces de encontrar una solución, en un tiempo de resolución inferior al $cutoff_2$. De igual manera, las

choice functions tampoco logran alcanzar soluciones en este mismo escenario.

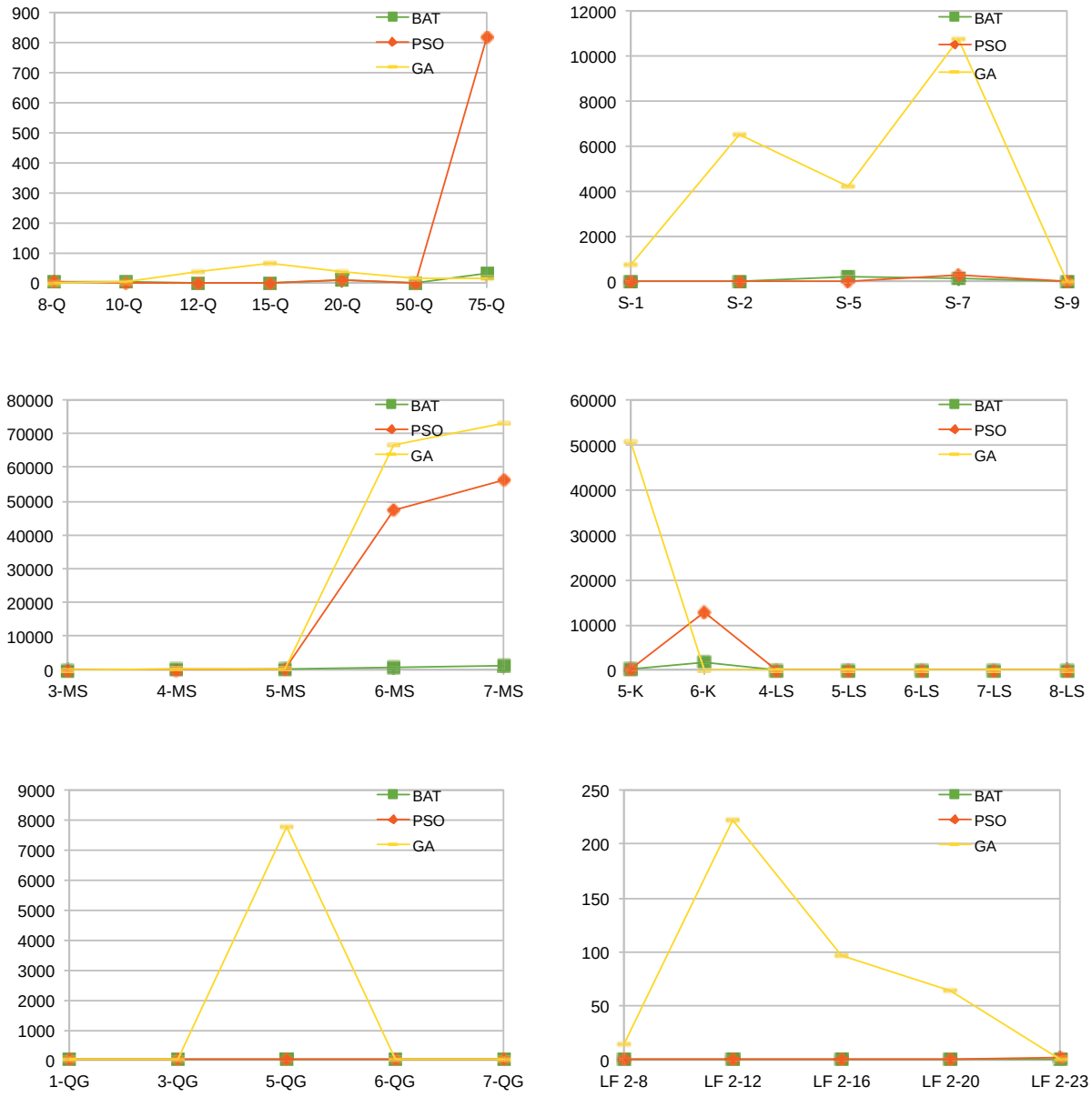

Figura 10: Comparación backtracks.

Por el contrario, la selección randomica es más rápida que BAT, pero este nuevo control en línea es ampliamente superior a PSO y GA (80 veces más rápido en promedio).

En las pruebas con las instancias del problema Magic Square (ver Tabla 9), en general las de mayor complejidad ($n > 4$), las estrategias de enumeración por sí solas, no logran resolverlas dentro de un contexto máximo de tiempo de resolución ($cutoff_2$), siendo inviable encontrar una solución bajo este escenario. Por su parte, las choice functions tampoco logran encontrar una solución bajo este contexto, aumentando en considerablemente el tiempo de resolución de cada instancia probada, demostrando con esto que son poco útiles para resolver estos problemas combinatoriales, en un tiempo menor de ejecución.

S_j	**Problemas**													
	N-Queens							$\bar{X}_Q$	**Sudoku**					$\bar{X}_S$
	8Q	**10Q**	**12Q**	**15Q**	**20Q**	**50Q**	**75Q**		**1S**	**2S**	**5S**	**7S**	**9S**	
S_1	5	5	12	57	20405	t.o.	t.o.	t.o.	5	35	7453	26882	4	8593,75
S_2	5	8	11	903	4867	t.o.	t.o.	t.o.	2247	30515	t.o.	t.o.	209	t.o.
S_3	5	3	11	3	16	532	4280	692,86	6	10	2181	2135	8	1083
S_4	4	4	11	59	20529	t.o.	t.o.	t.o.	6	50	8274	25486	5	8454
S_5	2	4	13	28	1294	t.o.	t.o.	t.o.	18	225	t.o.	t.o.	5	t.o.
S_6	4	5	14	79	26972	t.o.	t.o.	t.o.	6	1607	t.o.	t.o.	3	t.o.
S_7	4	3	11	3	15	524	4217	682,43	6	10	2247	2187	8	1112,5
S_8	2	4	10	24	93	t.o.	t.o.	t.o.	8	10	897	31732	8	8161,75
S_9	5	5	11	58	20349	t.o.	t.o.	t.o.	4	35	7521	26621	4	8545,25
S_{10}	5	8	11	971	4780	t.o.	t.o.	t.o.	4754	29797	t.o.	t.o.	215	t.o.
S_{11}	4	7	11	3	18	532	4336	701,57	6	10	2394	2069	8	1119,75
S_{12}	5	5	11	61	23860	t.o.	t.o.	t.o.	6	50	9015	26573	5	8911
S_{13}	2	5	13	29	1250	t.o.	t.o.	t.o.	18	225	t.o.	t.o.	4	t.o.
S_{14}	4	5	14	83	36034	t.o.	t.o.	t.o.	7	1732	t.o.	t.o.	3	t.o.
S_{15}	4	3	11	3	17	533	4195	680,86	6	10	2310	2094	8	1105
S_{16}	2	4	10	25	87	t.o.	t.o.	t.o.	8	10	972	30767	8	7939,25
S_{17}	5	4	11	58	22286	t.o.	t.o.	t.o.	8	5	t.o.	3725	6	t.o.
S_{18}	5	7	10	953	4547	t.o.	t.o.	t.o.	2497	18836	t.o.	t.o.	20	t.o.
S_{19}	4	2	11	3	16	520	4334	698,57	15	30	2590	338	7	743,25
S_{20}	4	4	11	59	13135	t.o.	t.o.	t.o.	15	5	t.o.	5350	8	t.o.
S_{21}	4	5	14	79	26515	t.o.	t.o.	t.o.	74	100	t.o.	t.o.	4	t.o.
S_{22}	2	4	13	28	1249	t.o.	t.o.	t.o.	3615	1710	t.o.	t.o.	4	t.o.
S_{23}	4	3	11	3	16	521	4187	677,86	15	30	2670	378	8	773,25
S_{24}	2	5	8	102	1528	t.o.	t.o.	t.o.	125	40	t.o.	9168	6	t.o.
CF_a	107	117	124	198	1449	t.o.	t.o.	t.o.	118	194	5487	t.o.	157	t.o.
CF_b	238	176	249	307	194	t.o.	t.o.	t.o.	173	321	t.o.	2587	158	1027
CF_c	159	141	159	179	3144	t.o.	t.o.	t.o.	258	277	379	t.o.	159	t.o.
CF_d	148	182	234	416	1613	475	2789	836,71	107	363	3569	15587	148	4906,5
CF_e	137	159	235	604	1797	1154	t.o.	t.o.	192	141	t.o.	395	116	t.o.
CF_f	142	175	215	542	2895	t.o.	t.o.	t.o.	166	124	t.o.	t.o.	159	t.o.
CF_g	116	155	313	441	2413	t.o.	t.o.	t.o.	146	214	t.o.	t.o.	113	t.o.
CF_h	146	161	289	558	589	t.o.	t.o.	t.o.	119	139	1109	t.o.	128	t.o.
CF_i	155	147	276	389	1367	t.o.	t.o.	t.o.	137	378	978	t.o.	131	t.o.
CF_j	125	151	187	487	617	625	2879	724,43	116	243	578	1128	111	516,25
CF_k	118	149	178	476	702	t.o.	t.o.	t.o.	132	139	t.o.	t.o.	147	t.o.
CF_l	119	156	194	143	2115	t.o.	t.o.	t.o.	114	152	t.o.	t.o.	160	t.o.
RND	4	2	6	10	71	16	55	23,43	10	6	156	578	8	187,5
PSO	4982	7735	24369	10483	52827	980195	997452	296863	12043	10967	979975	967014	10351	200465
GA	645	735	875	972	7520	6530	16069	4763,71	2270	15638	8202	25748	740	12964,5
BAT	**501**	**604**	**750**	**1024**	**1556**	**7856**	**20870**	**4737,29**	**651**	**586**	**628**	**593**	**608**	**614,5**

Tabla 8: Solving Time requerido para las diferentes estrategias de enumeración en los problemas N-Queens y Sudoku.

Las pruebas también demuestran que el nuevo enfoque BAT es más lento que la selección randómica, para las instancias pequeñas con $n \leqslant 5$, pero es mucho más eficiente en las instancias superiores $n \in \{6, 7\}$. En comparativa con los enfoques anteriores, BAT es notablemente superior a PSO y GA, logrando resolver

el 100 % de las instancias. En promedio, BAT supera a todas las técnicas de resolución.

Para las instancias del problema Latin Square (ver Tabla 9), las estrategias de enumeración y las choice functions, al contrario de los casos anteriores, logran resolverlas en su mayoría, bajo el entorno de minimizar el tiempo de resolución. Sin embargo, y a diferencia de lo ocurrido en la evaluación con el $cutoff_1$, los resultados obtenidos con el nuevo enfoque BAT no son tan determinantes. Como se puede apreciar en los resultados, existen casos en que las estrategias de enumeración y las choice functions, tienen un mejor desempeño que los enfoques propuestos. Pero cabe destacar que BAT, PSO y GA logran resolver la totalidad de las instancias del problema, por lo que en promedio, los controles en línea superan a las otras técnicas de resolución, siendo BAT el que menos tiempo utiliza.

S_j	**Problemas**											
	Magic Square					$\bar{X}_{MS}$	**Latin Square**					$\bar{X}_{LS}$
	3MS	**4MS**	**5MS**	**6MS**	**7MS**		**4LS**	**5LS**	**6LS**	**7LS**	**8LS**	
S_1	1	14	1544	t.o.	t.o.	t.o.	2	3	5	9	11	6
S_2	5	2340	t.o.	t.o.	t.o.	t.o.	2	11	102	t.o.	14	t.o.
S_3	1	6	296	t.o.	t.o.	t.o.	1	3	5	7	12	5,6
S_4	1	21	6490	t.o.	t.o.	t.o.	1	3	5	9	12	6
S_5	1	21	t.o.	t.o.	t.o.	t.o.	2	8	60	t.o.	14	t.o.
S_6	4	1500	t.o.	t.o.	t.o.	t.o.	2	4	7	88	12	22,6
S_7	1	6	203	t.o.	t.o.	t.o.	1	2	4	7	12	5,2
S_8	1	11	1669	t.o.	t.o.	t.o.	1	2	5	7	12	5,4
S_9	1	13	1498	t.o.	t.o.	t.o.	2	3	5	13	11	6,8
S_{10}	4	2366	t.o.	t.o.	t.o.	t.o.	1	11	103	t.o.	14	t.o.
S_{11}	1	6	297	t.o.	t.o.	t.o.	2	3	6	8	12	6,2
S_{12}	1	21	6053	t.o.	t.o.	t.o.	2	3	6	14	13	7,6
S_{13}	1	21	t.o.	t.o.	t.o.	t.o.	2	8	62	t.o.	15	t.o.
S_{14}	4	1495	t.o.	t.o.	t.o.	t.o.	2	4	7	92	14	23,8
S_{15}	1	6	216	t.o.	t.o.	t.o.	2	3	5	9	14	6,6
S_{16}	1	11	1690	t.o.	t.o.	t.o.	2	3	5	9	14	6,6
S_{17}	1	88	t.o.	t.o.	t.o.	t.o.	2	3	6	14	12	7,4
S_{18}	1	37	t.o.	t.o.	t.o.	t.o.	2	12	107	t.o.	14	t.o.
S_{19}	1	99	t.o.	t.o.	t.o.	t.o.	2	3	5	8	12	6
S_{20}	1	42	165878	t.o.	t.o.	t.o.	2	4	5	16	13	8
S_{21}	1	147	t.o.	t.o.	t.o.	t.o.	2	4	7	93	13	23,8
S_{22}	1	37	153679	t.o.	t.o.	t.o.	2	8	64	t.o.	15	t.o.
S_{23}	1	102	t.o.	t.o.	t.o.	t.o.	2	3	6	8	13	6,4
S_{24}	1	79	t.o.	t.o.	t.o.	t.o.	2	3	5	10	17	7,4
CF_a	118	165	216	t.o.	t.o.	t.o.	101	113	107	115	110	109,2
CF_b	106	277	t.o.	t.o.	t.o.	t.o.	129	155	113	114	162	134,6
CF_c	117	158	215	t.o.	t.o.	t.o.	207	112	182	113	115	145,8
CF_d	111	119	398	t.o.	t.o.	t.o.	155	113	157	118	112	131
CF_e	132	157	345	t.o.	t.o.	t.o.	146	149	208	174	185	172,4
CF_f	118	123	1598	t.o.	t.o.	t.o.	134	187	314	156	148	187,8
CF_g	114	166	1105	t.o.	t.o.	t.o.	117	145	137	t.o.	155	t.o.
CF_h	107	186	1209	t.o.	t.o.	t.o.	115	138	151	t.o.	121	t.o.
CF_i	109	207	t.o.	t.o.	t.o.	t.o.	118	119	307	112	105	152,2
CF_j	112	159	t.o.	t.o.	t.o.	t.o.	137	198	367	167	107	195,2
CF_k	109	162	t.o.	t.o.	t.o.	t.o.	109	127	116	115	119	117,2
CF_l	124	189	t.o.	t.o.	t.o.	t.o.	219	113	115	17176	112	3547
RND	2	9	40	t.o.	t.o.	t.o.	2	3	5	10	15	7
PSO	2745	15986	565155	t.o.	t.o.	t.o.	3323	6647	12716	20519	31500	14941
GA	735	1162	1087	t.o.	t.o.	t.o.	695	692	725	777	752	728,2
BAT	**545**	**902**	**775**	**1038**	**1189**	**889,8**	**333**	**387**	**438**	**496**	**560**	**442,8**

Tabla 9: Solving Time requeridos para las diferentes estrategias de enumeración en los problemas Magic Square y Latin Square.

Lograr alcanzar una solución a las instancias del problema combinatorial Knight's Tour, como se mencionó anteriormente, es un proceso que requiere de un arduo trabajo. De acuerdo a los resultados obtenidos (tabla 10), se puede observar que las estrategias de enumeración por sí solas, no son capaces de encontrar una solución, dentro del contexto donde el máximo de valor de tiempo de resolución, no supere el valor $cutoff_2$. Diferente es el caso de las choice functions, quienes al menos el 50 % de ellas, si logran alcanzar una solución, minimizando el tiempo de ejecución.

Nuevamente la pruebas demuestran que el nuevo control en linea BAT es superior a todas las técnicas de resolución, pues ninguna de ellas es capaz de resolver las instancias, bajo el contexto de no superar el valor de corte, para el tiempo de resolución ($cutoff_2$).

Para las instancias de menor complejidad del problema Quasigroup ($n < 5$), las estaregias de enumeración, al igual que las choice functions, logran encontrar una solución de forma eficiente, minimizando el tiempo de resolución requerido para el proceso (ver Tabla 10). Pero queda demostrado también que para las instancias de mayor complejidad ($n = 5$, por ejemplo), tanto las de estrategias de enumeración, como las choice functions no logran encontrar una solución en un tiempo de ejecución que no supere el valor de corte ($cutoff_2$). El nuevo enfoque BAT logra resolver todas las instancias, superando en promedio a las estrategias de enumeración, las choice functions, la selección randómica y los enfoques reportados GA y PSO.

Las pruebas realizadas a las diferentes instancias del problema Langford (ver Tabla 10), permiten inferir que encontrar una solución con sólo una estratregia de enumeración, en un tiempo de ejecución inferior al valor de corte es totalmente viable. Sin embargo existen instancias del problema (en general $\{2, 20\}$ y $\{2, 23\}$), en las que la estrategia de enumeración por sí sola no logra resolver el problema. Considerando las choice functions para resolver el problema Langford (ver Tabla 10), se demuestra que es viable encontrar una solución a cerca del 90 % de las instancias. Sin embargo, esto también demuestra que existe un 10 % de instancias que no logran ser resueltas, minimizando el tiempo de resolución utilizado.

Por último, teniendo en cuenta el tiempo de ejecución promedio utilizado para resolver los problemas combinatoriales, el nuevo enfoque BAT mantiene su primer lugar, por sobre GA y PSO, y siendo ampliamente superior a la selección randómica, a las estrategias de enumeración y las choice functions, lo que demuestra que si bien existen casos particulares en los que BAT no supera a los enfoques reportados anteriormente, en promedio, es muy superior a ellos, siendo una alternativa real y altamente competitiva que debe ser considerada.

S_j	Problemas														
	Knight's Tour		$\bar{X}_{KT}$	**Quasigroup**					$\bar{X}_{QG}$	**Langford** $L_{n=2}$					$\bar{X}_L$
	5KT	**6KT**		**1QG**	**3QG**	**5QG**	**6QG**	**7QG**		**8L**	**12L**	**16L**	**20L**	**23L**	
S_1	1825	90755	1705	1	1	t.o.	45	256	t.o.	3	20	70	191	79	72,6
S_2	t.o	t.o	t.o	1	1	t.o.	t.o.	8020	t.o.	10	242	70526	t.o.	t.o.	t.o.
S_3	2499	111200	56849,5	1	1	7510	15	10	2511,67	3	4	231	546	286	214
S_4	t.o	89854	t.o	1	1	t.o.	45	307	t.o.	4	29	115	318	140	121,2
S_5	t.o	t.o	t.o	1	1	t.o.	t.o.	943	t.o.	3	43	1217	61944	68254	26292,2
S_6	t.o	t.o	t.o	1	1	t.o.	3605	16896	t.o.	3	32	489	11	19	110,8
S_7	t.o	39728	t.o	1	1	9465	15	10	3163,33	3	4	237	553	285	216,4
S_8	t.o	t.o	t.o	2	1	t.o.	t.o.	16	t.o.	2	22	7	240	19	58
S_9	1908	93762	47835	2	1	t.o.	40	240	t.o.	3	20	69	185	79	71,2
S_{10}	t.o	t.o	t.o	1	1	t.o.	t.o.	13481	t.o.	10	270	55291	t.o.	t.o.	t.o.
S_{11}	2625	102387	52506	1	1	9219	15	10	3081,33	4	4	250	538	285	216,2
S_{12}	t.o	109157	t.o	1	1	t.o.	45	348	t.o.	4	29	118	312	140	120,6
S_{13}	t.o	t.o	t.o	1	1	t.o.	t.o.	1097	t.o.	3	44	1273	61345	71209	26774,8
S_{14}	t.o	t.o	t.o	1	1	t.o.	3565	18205	t.o.	3	32	530	11	19	119
S_{15}	t.o	46673	t.o	1	1	10010	15	11	3345,33	3	5	235	541	278	212,4
S_{16}	t.o	t.o	t.o	2	1	t.o.	t.o.	15	t.o.	2	21	8	237	19	57,4
S_{17}	1827	96666	49246,5	1	1	9743	7075	9	5609	3	18	66	170	75	66,4
S_{18}	t.o	t.o	t.o	1	1	t.o.	t.o.	1878	t.o.	11	242	55687	t.o.	t.o.	t.o.
S_{19}	2620	97388	50004	1	1	20	125	12	52,33	3	4	245	562	272	217,2
S_{20}	t.o	90938	t.o	1	1	10507	6945	9	5820,33	4	29	107	294	126	112
S_{21}	t.o	t.o	t.o	1	1	t.o.	t.o.	1705	t.o.	3	33	510	11	20	115,4
S_{22}	t.o	t.o	t.o	1	1	t.o.	t.o.	9	t.o.	3	43	1297	58732	73168	26648,6
S_{23}	t.o	40997	t.o	1	1	21	130	12	54,33	3	5	240	569	276	218,6
S_{24}	t.o	t.o	t.o	1	1	t.o.	t.o.	14	t.o.	5	13	584	15437	10	3209,8
CF_a	t.o.	t.o.	t.o.	108	121	t.o.	196	217	t.o.	118	138	375	14528	1031	3238
CF_b	t.o.	t.o.	t.o.	172	115	115	379	579	357,67	109	179	517	1878	t.o.	t.o.
CF_c	t.o.	t.o.	t.o.	157	112	t.o.	485	810	t.o.	164	141	2247	458	129	627,8
CF_d	t.o.	t.o.	t.o.	154	113	128	415	1548	697	147	311	875	5878	278	1497,8
CF_e	t.o.	t.o.	t.o.	156	112	t.o.	t.o.	336	t.o.	1183	136	457	815	117	541,6
CF_f	t.o.	t.o.	t.o.	157	111	t.o.	264	115	t.o.	154	189	212	554	113	244,4
CF_g	t.o.	t.o.	t.o.	148	166	t.o.	398	367	t.o.	139	387	315	854	t.o.	t.o.
CF_h	t.o.	t.o.	t.o.	113	114	t.o.	t.o.	137	t.o.	118	139	319	t.o.	121	t.o.
CF_i	t.o.	t.o.	t.o.	112	117	t.o.	1276	119	t.o.	278	120	771	428	876	494,6
CF_j	t.o.	t.o.	t.o.	106	111	t.o.	16587	413	t.o.	119	411	986	312	116	388,8
CF_k	t.o.	t.o.	t.o.	118	110	t.o.	786	131	t.o.	107	139	438	1201	765	530
CF_l	t.o.	t.o.	t.o.	112	107	t.o.	367	587	t.o.	129	118	298	886	1278	541,8
RND	70	56602	28336	2	10	15	13	19	15,67	3	10	70	7	10	20
PSO	21089	170325	95707	2160	1250	59158	44565	28612	44111,67	5000	10430	20548	28466	30468	18982,4
GA	4563751	20302204	12432977,5	675	665	11862	947	795	4534,67	762	1212	1502	1409	1287	1234,4
BAT	**3194**	**4570**	**3882**	**262**	**214**	**645**	**626**	**516**	**595,67**	**415**	**506**	**623**	**742**	**810**	**619,2**

Tabla 10: Solving Time requeridos para las diferentes estrategias de enumeración en los problemas Knight's Tour, Quasigroup y Langford.

Una comparación gráfica de los enfoques de control en línea, en relación al tiempo de resolución para las instancias, se ilustra en la Figura 11.

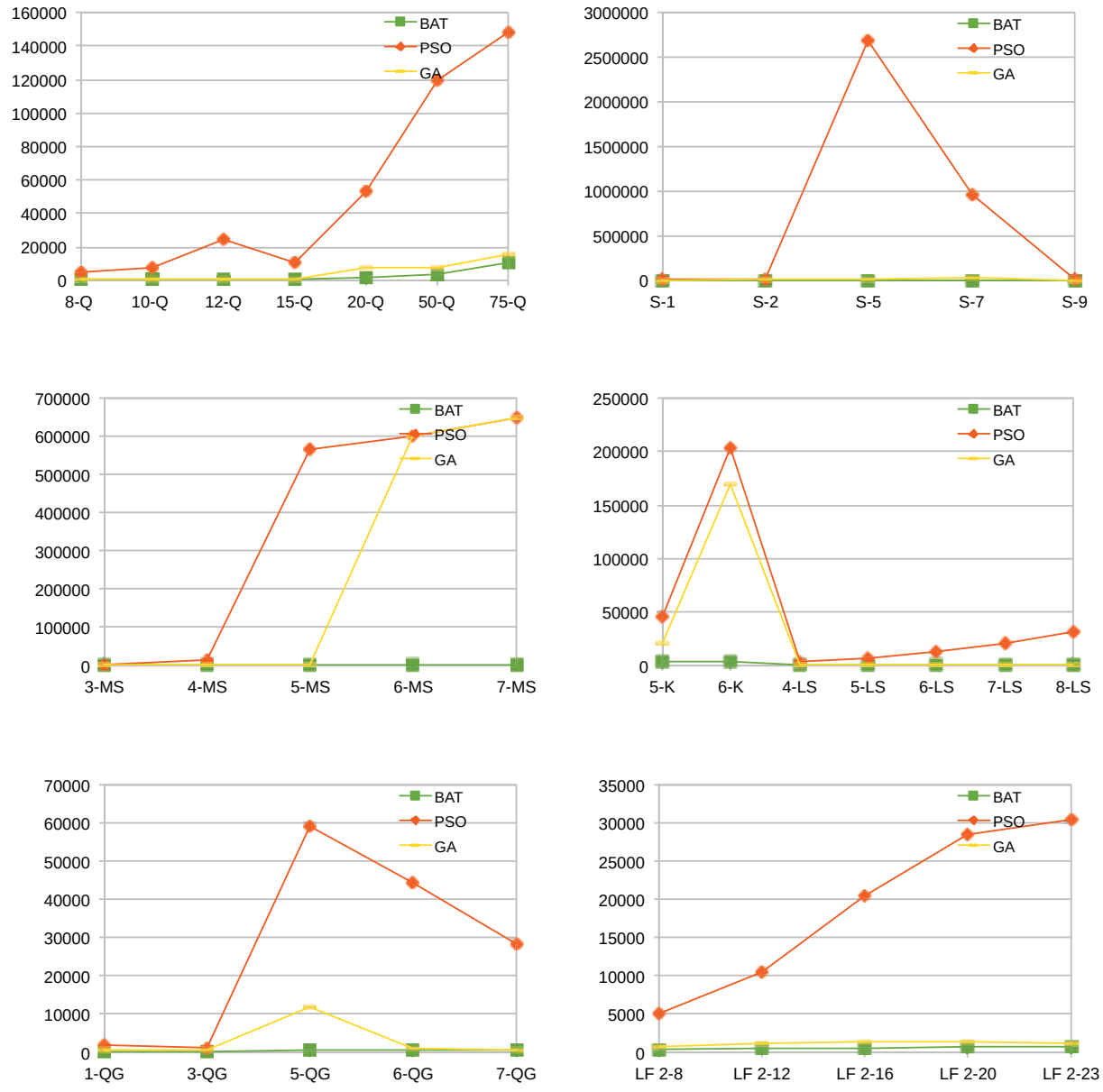

Figura 11: Comparación tiempo de resolución.

8. Conclusiones

En este trabajo, se ha presentado un nuevo enfoque para el control en línea basado en el optimizador Bat, para la solución de los CSP. La idea es intercalar un portafolio de veinticuatro estrategias de enumeración, con el fin de utilizar la más prometedora en cada paso del proceso.

La investigación sobre las características de la programación con restricciones, apoyó en gran medida la comprensión de este paradigma, recopilando información importante para la continuidad de todas las etapas del proyecto.

Al conocer la arquitectura de Autonomous Search, específicamente el funcionamiento del componente de *actualización*, permitió definir que era necesario asignar una importancia (o peso) a las estrategias en la *choice function*, ya que al incorporar el optimizador Bat, cada función de la estrategia donde se evalúan los indicadores, toma importancia a la hora de gestionar una posible solución. Por lo tanto es necesario utilizar en la resolución de un problema, un optimizador como el algoritmo Bat.

El control en línea se lleva a cabo por medio de un modelo de suma ponderada, que está finamente sintonizado mediante un algoritmo Bat. Los resultados obtenidos por el enfoque propuesto se pueden explicar por dos capacidades interesantes de algoritmos Bat: Zoom automático y de control de parámetro. La primera de ellas permite al algoritmo cambiar el zoom de forma automática dentro de las áreas del espacio de búsqueda, donde se han encontrado soluciones prometedoras, lo que resulta en tasas de convergencia rápidas. La segunda, permite la variación de la intensidad y la frecuencia de emisión de pulso (como se muestra en las ecuaciones 11 y 10) durante el tiempo de la solución, esto le permite cambiar automáticamente desde la exploración hasta la explotación cuando se acerque la solución óptima. Los beneficios de ambas características pueden particularmente, ser observado en la resolución de los casos más complejos de CSP (50-Q, 75-Q, 6-MS, 7-MS, 5-KT, 6-KT y 5-QG), donde los tiempos de ejecución son notablemente mejor en comparación con GA y PSO.

Los resultados obtenidos de los experimentos abren oportunidades para la investigación futura. En el mediano plazo, se tiene la intención de incorporar nuevas y más sofisticadas estrategias de enumeración, así como para experimentar con nuevas metaheurísticas modernas para la optimización. Otra idea interesante es sobre el diseño de framework similar para intercalar diferentes técnicas de filtrado de dominio.

A. Anexos

A.1. Indicadores

A.1.1. Lista de indicadores base

Código	Nombre	Descripción	Cálculo
Step	Número de mediciones	Cada vez que se utiliza la *choice function*.	$step++$
TV	Número total de variables	Número total de variables en el problema. Indicador que guarda relación directa con las caraterísticas del problema. Este valor es constante durante el proceso.	$length(AllVars, N)$
VU	Variables no instanciadas	Número de varibles que no han sido instanciadas (utilizadas) en el proceso.	$var\_count(Rest, VU)$
SB	Shallow backtrack	Intento fallido de asignar un valor a una variable. Se vuelve a intentar asignar, pero con el valor siguiente.	–
SS	Search Space	Espacio actual de búsqueda	$\prod(Dom_i)_t$
$\mathbf{SS}_{pr}$	Previous Search Space	Espacio anterior de búsqueda	$\prod(Dom_i)_{t-1}$
B	Backtracks	Intento fallido de solución por parte de una variable. Se debe retroceder a la variable anterior.	$backtrack++$
N	Nodos	Cantidad de nodos visitados	$N++$

Tabla 11: Indicadores base

A.1.2. Lista de indicadores calculados

Código	Nombre	Descripción	Cálculo
VF	Variables instanciadas	Número de variables instanciadas por enumeración y propagación.	$TV - VU$
VFE	Variables fijadas por enumeración en cada paso	Opera cuando el número de pasos es mayor que 1 ($step > 1$).	l
TVFE	Número total de VFE	Número total de variables instanciadas.	$\sum VFE$
VFP	Variables fijadas por propagación en cada paso	Número de variables instanciadas por propagación.	$length(Rest, N) - VU$
TVFP	Número total de VFP	Número total de variables instanciadas por propagación.	$\sum VFP$
TSB	Total Shallow Backtracks	Número total de Shallow Backtracks	$\sum SB$
$\mathbf{d}_{pr}$	Profundidad previa	Profundidad previa en el árbol de búsqueda.	$(TV - VU)_{t-1}$
D	Profundidad	Profundidad actual en el árbol de búsqueda.	$(TV - VU)_t$
$\mathbf{dmax}_{pr}$	Profundidad máxima previa	La profundidad máxima previa en el árbol de búsqueda.	$MAX(dmax)_{n-1}$
In1	Profundidad máxima actual - dmax_{pr}	Representa la variación de la profundidad.	$(dmax - dmax_{pr})$
In2	Profundidad actual - d_{pr}	Si es positivo, el nodo actual se encuentra a más profundidad que el del paso previo.	$(d_t - d_{t-1})$
SSR	Reducción del espacio de búsqueda	Si es positivo, el actual espacio de búsqueda es más pequeño que el del *snapshot* anterior.	$100\ \frac{(SS_{pr} - SS)}{SS_{pr}}$
PVFE	Porcentaje de VFE	–	$100\ \frac{VFE}{TV}$
PVFET	Porcentaje de TVFE	–	$100\ \frac{TVFE}{TV}$
PVFP	Porcentaje de VFP	–	$100\ \frac{VFP}{TV}$
PVFPT	Porcentaje de TVFP	–	$100\ \frac{TVFP}{TV}$
Thrash	Thrashing	–	$d_{t-1} - VFPS_{t-1}$
$\mathbf{B}_{real}$	–	Si la variable actual lleva a una ruta sin salida, entonces se va a la variable previa y cuenta ese retroceso	Si $\text{D} - \text{d}_{pr} == 0$, entonces $\text{B}_{real} + +$ Si $D < \text{d}_{pr}$, entonces $\text{d}_{pr} - \text{D} + 1$

Tabla 12: Indicadores calculados

A.2. Modelos ECL^iPS^e

A.2.1. N-Queens

El algoritmo 2, implementa el modelo asociado al problema de las N-Reinas, desarrollado en ECL^iPS^e.

Algoritmo 2 Algoritmo para el prolema N-Reinas en ECL^iPS^e

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

queens(N) :-
    queens(N, Board),
    lib_autonomous_search(Board).

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

A.2.2. Sudoku puzzle

El algoritmo 3, implementa el modelo asociado al problema Sudoku Puzzle, desarrollado en ECL^iPS^e.

Algoritmo 3 Algoritmo para el prolema Sudoku en ECL^iPS^e

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

sudoku(ProblemName) :-
    problem(ProblemName, Board),
    sudoku2(3, Board).

sudoku2(N, Board) :-
    N2 is N*N,
    dim(Board, [N2,N2]), Board[1..N2,1..N2] :: 1..N2,
    ( for(I,1,N2), param(Board,N2) do
        Row is Board[I,1..N2],
        alldifferent(Row),
        Col is Board[1..N2,I],
        alldifferent(Col)
    ),
    ( multifor([I,J],1,N2,N), param(Board,N) do
        ( multifor([K,L],0,N-1), param(Board,I,J), foreach(X,SubSquare) do
            X is Board[I+K,J+L]
        ),
    alldifferent(SubSquare)
),
term_variables(Board, Vars).
```

A.2.3. Magic Squares

El algoritmo 4, implementa el modelo asociado al problema Magic Squares, desarrollado en ECL^iPS^e.

Algoritmo 4 Algoritmo para el prolema Magic Squares en ECL^iPS^e.

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

magic(N):-
    magic2(N,Board,Vars),
    lib_autonomous_search(Board).

magic2(N,Vars,Square):-
    NN is N*N,
    Sum is N*(NN+1)//2,
    dim(Square, [N,N]),
    Square[1..N,1..N] :: 1..NN,
    Rows is Square[1..N,1..N],
    flatten(Rows, Vars),
    alldifferent(Vars),

( for(I,1,N), foreach(U,UpDiag), foreach(D,DownDiag), param(N,Square,Sum) do
    Sum #= sum(Square[I,1..N]),
    Sum #= sum(Square[1..N,I]),
    U is Square[I,I],
    D is Square[I,N+1-I]
),
Sum #= sum(UpDiag),
Sum #= sum(DownDiag),
Square[1,1] #< Square[1,N],
Square[1,1] #< Square[N,N],
Square[1,1] #< Square[N,1],
Square[1,N] #< Square[N,1].
```

A.2.4. Knight's Tour

El algoritmo 5, implementa el modelo asociado al problema Knight's Tour, desarrollado en ECL^iPS^e.

Algoritmo 5 Algoritmo para el prolema Knight Tour en ECL^iPS^e.

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

knightTour(N):-
    knight_tour(N, Board),
    lib_autonomous_search(Board).

knight_tour(N, Board):-
    N2 is N*N,
    length(Board, N2),
    C = [-2,-1,1,2],
    ic_global:alldifferent(Board),
    ( fromto(Board, [Q1|Cols], Cols, []),count(Dist,1,_), param(N,C) do
        tomar_Q2(Cols,Q2),
        I :: 1..N,
        J :: 1..N,
        A :: C,
        B :: C,
        IA #= I+A,
        JB #= J+B,
        IA #>= 1,
        JB #>= 1,
        IA #=< N,
        JB #=< N,
        abs(A)+abs(B) #= 3,
        (I-1)*N + J #= Q1,
        Q2 #= (I-1+A)*N + (J+B)
    ).
    tomar_Q2([Q2|Cols],Q2).
    tomar_Q2([],Q2).
```

A.2.5. Latin Squares

El algoritmo 6, implementa el modelo asociado al problema Latin Square, desarrollado en ECL^iPS^e.

Algoritmo 6 Algoritmo para el problema Latin Square en ECL^iPS^e.

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

latin_square(N):-
    latin_square(N,Vars),
    term_variables(Vars, Board),
    init_globals,
    Board2 = Board,
    seTssTotal(Board),
    lab(Board,Board2),
    end_java(Board).

latin_square(N, X):-
    dim(X, [N,N]),
    X[1..N,1..N] :: 1..N,
    (
        for(I, 1, N),
        param(X, N)
        do
            alldifferent(X[1..N, I]),
            alldifferent(X[I, 1..N])
    ).
```

A.2.6. Langford

El algoritmo 7, implementa el modelo asociado al problema Langford, desarrollado en ECL^iPS^e.

Algoritmo 7 Algoritmo para el problema Langford en ECL^iPS^e.

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

langford(Problem) :-
    funcion_retorno_lf(Problem,N,K),
    K2 is N*K,
    length(Position, K2),
    Position :: 1..K2,

    length(Solution,K2),
    Solution :: 1..K,

    alldifferent(Position),

    nth1(1,Solution,Solution1),
    nth1(K2,Solution,SolutionK2),
    Solution1 #< SolutionK2,

    (for(I,1,K), param(Position,K) do
        IK is I+K,
        nth1(IK, Position, PositionIK),
        nth1(I, Position, PositionI),
        I1 is I+1,
        PositionIK #= PositionI + I1,
        SolutionPositionI #= I,
        SolutionPositionIK #= I
    ),
    term_variables(Position, Vars),
    init_globals,
    Vars2 = Vars,
    lab(Vars,Vars2),
    end_java(Position).

    funcion_retorno_lf(1,2,8).
    funcion_retorno_lf(2,2,12).
    funcion_retorno_lf(3,2,16).
    funcion_retorno_lf(4,2,20).
    funcion_retorno_lf(5,2,23).
```

A.2.7. Quasigroup

El algoritmo 8, implementa el modelo asociado al problema Quasigroup, desarrollado en ECL^iPS^e.

Algoritmo 8 Algoritmo Quasi Group en ECL^iPS^e.

```
\begin{verbatim}
:-lib(ic).
:-lib(lists).
:-lib(ic_global).
:-lib(random).
:-lib(lib_autonomous_search).

quasigroup(ProblemName):-
    problem_qg(ProblemName, Board),
    length(ProblemName,N),
    quasigroup(Board,N).

quasigroup(Board,N) :-
    dim(Board, [N,N]),
    Board[1..N,1..N] :: 1..N,
    (for(I, 1, N), param(Board, N) do
        alldifferent(Board[I,1..N]),
        alldifferent(Board[1..N,I])
    ),
    term_variables(Board, Vars),
    init_globals,
    Vars2 = Vars,
    lab(Vars,Vars2),
    end_java(Board).

    length(1,5).
    length(2,4).
    length(3,4).
    length(4,4).
    length(5,10).
    length(6,10).
    length(7,10).
    length(8,10).
    length(9,10).
```

Referencias

[1] Broderick Crawford, Carlos Castro, Eric Monfroy. *Programación con restricciones dinámicas.* 2009.

[2] Francesca Rossi, Peter van Beek, Toby Walsh. *Handbook of Constraint Programming.* 2006.

[3] Federico Barber, Miguel A. Salido *Introducción a la programación de restricciones.* Disponible vía web en http://users.dsic.upv.es/ msalido/papers/aepia-introduccion.pdf.

[4] Krzysztof Apt. *Principles of Constraint Programming.* Cambridge Press, 2003.

[5] Ricardo Soto, Hakan Kjellerstrand, Juan Gutierrez, Alexis Lopez, Broderick Crawford, and Eric Monfroy. *Solving Manufacturing Cell Design Problems Using Constraint Programming.* In proceedings of the 25th International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems (IEA/AIE), pp. 400-406, Springer, 2012.

[6] Renzo Pizarro, Gianni Rivera. *Modelado y resolución del Nurse Rostering Problem (NRP) utilizando programación con restricciones: un caso de estudio.* Informe Final para optar al título de Ingeniero de Ejecución Informática. Pontificia Universidad Católica de Valparaíso, 2011.

[7] Kam Pui Chow. *Airport counter allocation using constraint logic programming.* In Proc. of Practical Application of Constraint Technology, 19

[8] Ivan Edward Sutherland. *Sketchpad: A man-machine graphical communication system.* Technical Report, Computer Laboratory, University of Cambridge, 2003.

[9] Krzysztof Apt, Mark Wallace. *Constraint Logic Programming using Eclipse.* 2006.

[10] Krzysztof Apt, Jacob Brunekreef, Vincent Partington, and Andrea Schaerf. Alma-0: An Imperative Language that Supports Declarative Programming. *ACM Transactions on Programming Languages and Systems (ACM TOPLAS)*, 20(5):1014–1066, 1998.

[11] Broderick Crawford, Carlos Castro, Eric Mofroy, Ricardo Soto, Wenceslao Palma and Fernando Paredes. *Dynamic Selection of Enumeration Strategies for Solving Constraint Satisfaction Problems.* Romanian Journal of Information Science And Technology Vol.15, pp. 106-128, 2012

[12] Broderick Crawford, Carlos Castro, Eric Monfroy, Ricardo Soto, Wenceslao Palma, and Fernando Paredes. *Hyperheuristic Approach for Guiding Enumeration in Constraint Solving.* EVOLVE - A Bridge between Probability, SetOriented Numerics, and Evolutionary Computation II, Advances in Intelligent Systems and Computing (AISC), Vol.175, pp. 171-188, Springer, 2013.

[13] A. E. Eiben, S. K. Smit, 2012. Autonomous Search. Springer, Ch. Evolutionary Algorithm Parameters and Methods to Tune Them.

[14] Jorge Maturana, Álvaro Fialho, Frédéric Saubion, Marc Schoenauer, Frédéric Lardeux and Michéle Sebag. Autonomous Search. Springer, Ch. 2012. Adaptive Operator Selection and Management in Evolutionary Algorithms.

[15] Jorge Maturana, Frédéric Lardeux, Frédéric Saubion, 2010. Autonomous operator management for evolutionary algorithms. J. Heuristics 16 (6), 881-909.

[16] Jorge Maturana, Frédéric Saubion On the design of adaptive control strategies for evolutionary algorithms. In: Proceedings of the 8th International Conference on Artificial Evolution (EA). Vol. 4926 of LNCS, 2007 Springer, pp. 303-315.

[17] Jorge Maturana, Frédéric Saubion, 2008. A compass to guide genetic algorithms. In: Proceedings of the 10th International Conference on Parallel Problem Solving from Nature (PPSN). Vol. 5199 of LNCS. Springer, pp. 256-265.

[18] Thomas Stützle, Manuel López-Ibáñez, Paola Pellegrini, Michael Maur, Marco Montes de Oca, Mauro Birattari, Marco Dorigo. Parameter Adaptation in Ant Colony Optimization Autonomous Search. Springer, Ch. 2012.

[19] Frank Hutter, Domagoj Babic, Hoos Holger, AJ Hu. Boosting verification by automatic tuning of decision procedures. In: In Proceedings of the 7th International Conference on Formal Methods in Computer-Aided Design (FMCAD). IEEE Computer Society, pp. 27-34, 2007

[20] Frank Hutter, Youssef Hamadi, Holger Hoos, Kevin Leyton-Brown, 2006. Performance prediction and automated tuning of randomized and parametric algorithms. In: CP. pp. 213-228.

[21] Yuri Malitsky, Meinolf Sellmann, 2012. Instance-specific algorithm configuration as a method for non-model-based portfolio generation. In: Proceedings of the 9th International Conference on the Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR). Vol. 7298 of LNCS. Springer, pp. 244-259.

[22] Lin Xu, Frank Hutter, Holger Hoos, Kevin Leyton-Brown, 2008. Satzilla: Portfolio-based algorithm selection for SAT. J. Artif. Intell. Res. (JAIR) 32, 565-606.

[23] Frank Hutter, Holger Hoos, Kevin Leyton-Brown, 2010. Automated configuration of mixed integer programming solvers. In: Proceedings of the 7th International Conference on the Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR). Vol. 6140 of LNCS. Springer, pp. 186-202.

[24] Frank Hutter, Holger Hoos, Kevin Leyton-Brown, 2011. Sequential model-based optimization for general algorithm configuration. In: In Proceedings of the 5th International Conference on Learning and Intelligent Optimization (LION). Vol. 6683 of LNCS. Springer, pp. 507-523.

[25] Susan Epstein, Smiljana Petrovic, 2007. Learning to solve constraint problems. In: Proceedings of the Workshop on Planning and Learning (ICAPS).

[26] Susan Epstein, Eugene Freuder, Richard Wallace, Anton Morozov, Bruce Samuels, 2002. The adaptive constraint engine. In: Proceedings of the 8th International Conference on Principles and Practice of Constraint Programming (CP). Vol. 2470 of Lecture Notes in Computer Science. Springer, pp. 525-542.

[27] Yuehua Xu, David Stern, Horst Samulowitz, 2009. Learning adaptation to solve constraint satisfaction problems. In: Proceedings of the 3rd International Conference on Learning and Intelligent Optimization (LION). pp. 507-523.

[28] Frédéric Boussemart, Fred Hemery, Christophe Lecoutre, Lakhdar Sais, 2004. Boosting systematic search by weighting constraints. In: Proceedings of the 16th Eureopean Conference on Artificial Intelligence (ECAI). IOS Press, pp. 146-150.

[29] Diarmuid Grimes, Richard Wallace, 2007. Learning to identify global bottlenecks in constraint satisfaction search. In: Proceedings of the Twentieth International Florida Artificial Intelligence Research Society (FLAIRS) Conference. AAAI Press, pp. 592-597.

[30] Richard Wallace, Diarmuid Grimes, 2008. Experimental studies of variable selection strategies based on constraint weights. J. Algorithms 63 (1-3), 114-129.

[31] Youssef Hamadi, Eric Monfroy, Frédéric Saubion. *What is Autonomous Search?.* Springer, 2008.

[32] Vćtor Bustos. *Una arquitectura modular para autonomous search. Informe Final para optar al título de Ingeniero Civil en Informática, Pontificia Universidad Católica de Valparaíso.* 2012.

[33] Ricardo Soto, Broderick Crawford, Eric Monfroy, and Víctor Bustos. *Using Autonomous Search for Generating Good Enumeration Strategy Blends in Constraint Programming.* In proceedings of the 12th International Conference on Computational Science and Its Applications (ICCSA), pp. 607-617, Springer, 2012.

[34] Broderick Crawford, Ricardo Soto, Mauricio Montecinos, Carlos Castro, and Eric. Monfroy. *A Framework for Autonomous Search in the Eclipse Solver.* Proceedings of the 24th International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems (IEA/AIE), páginas 79-84, LNCS 6703, Springer, 2011.

[35] Broderick Crawford, Ricardo Soto, Eric. Monfroy, Wenceslao Palma, Carlos Castro, Fernando Paredes *Parameter tuning of a choice-function based hyperheuristic using Particle Swarm Optimization.* Expert Systems with Applications, vol 40(5), páginas 1690 - 1695, 2013.

[36] Xin-She Yang. *Bat Algorithm: Literature Review and Applications.* Bio-Inspired Computation, Vol. 5, No. 3, pp. 141 - 149 (2013).

[37] Rodrigo Yuji Mizobe Nakamura, Luís Pereira, David Rodríguez, JP Papa, Xin-She Yang. *BBA: A binary bat algorithm for feature selection.* 25th SIBGRAPI Conference on Graphics, Patterns and Images (SIBGRAPI), 22 - 25 Aug. 2012, IEEE Publication, pp. 291 - 297.

[38] Xin-She Yang. *Bat algorithm and cuckoo search: a tutorial.* Artificial Intelligence, Evolutionary Computing and Metaheuristics (Eds. X. S. Yang), Studies in Computational Intelligence, Vol. 427, pp. 421 - 434