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

ENUMERACION ADAPTATIVA Y RANKEADORES DE ESTRATEGIAS BASADOS EN ALGORITMOS DE BÚSQUEDA GRAVITACIONAL

RODRIGO EDUARDO HERRERA LAFERTE

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 ALGORITMOS DE BÚSQUEDA GRAVITACIONAL

RODRIGO EDUARDO HERRERA LAFERTE

Profesor guía: **RICARDO SOTO DE GIORGIS**

Abril, 2015

Dedicada

A mi esposa Denisse Hinojosa por su apoyo

incondicional, a mi familia, amigos y por

su gran compañerismo a Rodrigo "Vita" Olivares.

Gracias totales...

Resumen

El proyecto tiene como objetivo principal mejorar la eficiencia de la resolución de problemas de satisfacción de restricciones. Para conseguir esto se utilizará Autonomous Search (AS), técnica que entrega características de auto-ajuste a los sistema de resolución. En particular, puede cambiar en tiempo real las estrategias de enumeración que presenten un bajo desempeño por otras que sean más eficientes. Para realizar esto se utilizará una función de selección, la cual se encarga de rankear las estrategias de enumeración en base a ciertos indicadores. De esta manera, se asignará en cada iteración un valor de importancia a cada estrategia dependiendo del comportamiento que esta tenga. Para potenciar este rankeo, se implementará una metaheurística llamada Gravitational Search Algorithm. Esta metaheurística que se basa en las leyes Newtonianas de gravedad y movimiento, pretende ayudar a la función de selección en la asignación de los valores de cada indicador para poder determinar los grados de importancia de las distintas estrategias. Con esto se pretende encontrar resultados que demuestren una mejoría en el desempeño de Autonomous Search.

Abstract

The main goal of this project is to improve the efficiency of the constraint satisfaction problem resolution. To this end, we employ Autonomous Search (AS) which is a technique that provides self-adjustment features to resolution systems. In particular, it is able to replace enumeration strategies presenting poor performances by other more efficient ones. To carry out this we employ a choice function, which is responsible to rank the enumeration strategies based on different indicators. In this way, we assign an importance value at each iteration based on the behavior of the strategy. To enhance this ranking we implement a metaheuristic called Gravitational Search Algorithm. This metaheuristic, based on the Newtonian laws of gravity and motion, aims to help the Choice Function in the configuration of the indicator values to determine the degrees of importance of the different enumeration strategies. The idea is to finding results that demonstrate an improvement of the Autonomous Search performance.

Palabras clave: Autonomous Search, adaptativos, estrategias de enumeración, función de selección, Choice Function, rankear, iteración, metaheurística, Gravitational Search Algorithm, Newtonianas.

Contenido

Lista de Figuras

Lista de Tablas

Lista de Algoritmos

1 Introducción

La programación con Restricciones (CP), es un paradigma de la programación, donde las relaciones entre las variables son expresadas en términos de restricciones y que actualmente se utiliza para la resolución de problemas de optimización y de satisfacción de restricciones. CP es utilizado para describir y resolver problemas complejos en diversas áreas de estudio, tales como la biología molecular, aplicaciones de negocio, ingeniería eléctrica y la álgebra computacional entre otros [41]. Estos problemas pueden modelarse como problemas de satisfacción de restricciones (Constraint Satisfaction Problems - CSP) y resolverse usando técnicas de satisfacción de restricciones. Los problemas de CP son representados a través de variables, las cuales se expresan por medio de restricciones y dominios. La idea principal es buscar un estado en donde cada una de las restricciones de estas variables sean satisfechas simultáneamente.

Un problema de CP abarca un gran árbol de variables y puede comprender una enorme cantidad de restricciones. Lo que hace el programa es buscar y encontrar los valores para cada una de las variables basándose en las restricciones que limitan sus dominios. La búsqueda de los valores para cada una de estas variables es un proceso muy costoso en cuanto a tiempo y procesamiento, es por esto que AS se encarga de adaptar los procesos de búsqueda cuando estos presenten un bajo rendimiento en la resolución de un determinado problema. La labor que realiza AS es básicamente evaluar y clasificar las heurísticas a través del tiempo, cuando estas muestren un bajo rendimiento entonces deben ser remplazadas en tiempo real de ejecución. Las heurísticas se sub dividen en heurísticas de selección de variable y heurísticas de selección de valor. Cuando se agrupa una heurística de selección de valor con una de selección de variable se forma la llamada estrategia de enumeración. La forma en que se recorre el árbol de búsqueda para encontrar la solución es determinada por esta estrategia de enumeración. AS realiza el rankeo de cada una de estas estrategias a través de una función de selección, la cual evalúa cada estrategia en base a un conjunto de indicadores y parámetros de control que determinan la relevancia del indicador.

En el presente trabajo se busca potenciar el trabajo de la función de selección incorporando un nuevo optimizador basado en la metaheurística Gravitational Search Algorithm (GSA), la cual está inspirada en las leyes Newtonianas de movimiento y atracción. Este optimizador se encargará de asignar los valores a los parámetros de control de los indicadores para poder determinar el grado de importancia de cada estrategia. El solver que utilizaremos para resolver los problemas y evaluar las soluciones será ECLiPSe.

2 Objetivos

2.1 Objetivo General

Implementar un nuevo optimizador (GSA) para Autonomous Search junto con nuevas estrategias de enumeración para poder realizar experimentos y comparaciones de los resultados obtenidos.

2.2 Objetivo Específico

- Investigar y comprender el contexto de la programación con restricciones y su metodología para la resolución de problemas.
- Analizar la arquitectura de Autonomous Search.
- Implementar el optimizador Gravitational Search Algorithm.
- Implementar nuevas estrategias de enumeración.
- Implementar nuevas instancias y nuevos problemas.
- Realizar los diferentes experimentos y analizar los resultados obtenidos.

3 Estado del Arte

Autonomous Search es un enfoque moderno que permite al solver adaptar sus propios parámetros y heurísticas para de esta manera 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 [23, 24, 25, 26, 27, 28].

En los dominios de búsqueda completa, ambos métodos también han participado fuertemente, por ejemplo para impulsar con éxito SAT [29, 30, 31, 32] y solvers de programación entera mixta [33, 34] 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 [35, 36] y [37]. Otra idea tras el mismo objetivo propone asociar pesos a restricciones [38], 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 [39, 40], 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

La programación con restricciones es el estudio de sistemas computacionales basados en restricciones, que se utilizan para describir y posteriormente resolver problemas generalmente de tipo combinatorio, como también de optimización por medio de la declaración de restricciones en el dominio del problema con el objetivo de encontrar soluciones que cumplan con todas estas.

Los inicios de CP datan de las décadas del '60 y '70, en el ámbito de la inteligencia artificial para el entendimiento del lenguaje natural. A través del tiempo se ha aplicado en diversas áreas del conocimiento como la investigación operativa, la planificación, bases de datos, sistema de recuperación de la información, etc. Entre sus principales campos de aplicación se encuentran por ejemplo: análisis financiero, diseño y construcción de circuitos integrados, la resolución de problemas geométricos, asignación de stands en los aeropuertos o asignación de pasillos de salida en aeropuertos de Hong Kong [13], desarrollo de turnos de enfermeras de un hospital [10].

Para solucionar los ejemplos anteriores, CP se puede utilizar como un problema de optimización o de satisfacción. Ambas comparten las mismas terminologías pero sus técnicas de resolución son distintas. Los problemas de optimización se resuelven como un problema de satisfacción regular, pero el objetivo es generar una solución óptima a través de una función objetivo.

En cambio, los problemas de satisfacción de restricciones (Constraint Satisfaction Problem, CSP) constan de conjuntos finitos de variables, con un dominio de valores para cada una de estas y un conjunto de restricciones que acotan la combinación de valores que las variables pueden tomar. De esta forma se puede encontrar un valor para cada variable que satisfaga todas las restricciones impuestas en el problema [14], esto quiere decir que se pueden encontrar muchas soluciones siempre y cuando respeten las restricciones.

Cada una de estas formas de CP tiene como fases de resolución el modelado y la búsqueda. En el modelado se debe analizar el problema, luego debe ser planteado como un CSP y por último se debe implementar el modelo en un lenguaje para la programación con restricciones. La búsqueda es parte de las técnicas de resolución la cual se lleva a cabo a través de algoritmos de búsqueda, que se encargan de buscar en un espacio definido las posibles asignaciones de valores a las variables considerando las restricciones del problema, con el fin de encontrar la solución ó en el caso contrario, demostrar que no existe.

Para ayudar a mejorar la eficiencia de estos algorítmos se utilizan las técnicas de consistencia, las cuales verifican la consistencia entre variables actuales y futuras. Se utiliza además una heurística de selección de variable y valor, que establece el orden para el estudio de éstas, mejorando considerablemente la eficiencia del tiempo de resolución.

Por último CP cuenta con la ventaja de que el uso de las restricciones puede disminuir eficientemente el espacio de búsqueda, eliminando combinaciones de valores de variables que no cumplen con las condiciones antes mencionadas o simplemente no pueden aparecer juntas.

4.1 Problema de Satisfacción de Restricciones

En la actualidad hay muchos problemas que pueden ser resueltos como un problema de satisfacción de restricciones. Un CSP se compone de las variables que el problema requiere definir y el dominio de cada una de éstas, ambas deben pertenecer a un conjunto finito, y la o las restricciones que limitarán el conjunto de asignaciones para las variables implicadas, necesarias para resolver el problema. Esta solución, si es que existe, será la asignación de un valor del dominio a cada variable de forma que se cumpla cada una de las restricciones. La resolución de un problema de satisfacción de restricciones se compone de dos partes; la primera es el modelado el cual corresponde a la definición de las variables, dominios y restricciones, y la segunda parte es la resolución donde se utilizarán los algoritmos de búsqueda que según su propio método de funcionamiento encontrarán la solución o de lo contrario probará que esta no existe.

4.1.1 Definición de un CSP

Un CSP se representa como una terna (X, D, C) donde:

- X es un conjunto de variables $x_1, x_2, ..., x_n$.
- D es un conjunto de dominios $d_1, d_2, ..., d_n$. donde d_i es el dominio de x_i, e $i = 1, ..., n$.
- C es el conjunto finito de restricciones $c_1, c_2, ..., c_m$, donde $j = 1, ..., m$. tal que c_j es la relación sobre el conjunto de variables $\{x_1, x_2, ..., x_n\}$ y $c(x_i, x_j)$ son restricciones entre x_i y x_j restringiendo los valores que pueden tomar las variables.

En un CSP existen diferentes formas de modelar un problema para llegar a su resolución, debido a esto a continuación se detallan 2 ejemplos de modelado:

4.1.1.1 N-reinas

Este problema consiste en colocar N reinas en un tablero de ajedrez de tamaño NxN, de tal manera que no se amenacen entre ellas, una reina amenaza a otra si está en la misma fila, columna o diagonal. Se considerará para el siguiente ejemplo cantidad de reinas $N = 4$.

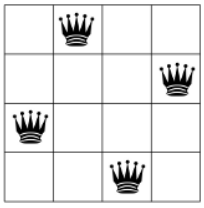

Figura 1: Solución al problema 4-reinas.

Modelo:

- Definición de las variables que en este caso corresponden a las 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 $j \in [i+1,4]$.

No colocar las reinas en la diagonal noroeste, sureste: $R_i - R_j \neq j - i$. No colocar las reinas en la diagonal suroeste, noreste: $R_i - R_j \neq i - j$.

La figura 1 muestra la disposición sobre el tablero de 4 reinas sin que se puedan atacar, satisfaciendo cada una de las restricciones, por lo tanto corresponde a una solución, ya que cumple las restricciones.

4.1.1.2 Criptográfico

En este ejemplo (figura 2) se presenta un problema criptográfico, en el cual se plantea una ecuación en la que se deben remplazar las letras *{s,e,n,d,m,o,r,y}* por dígitos distintos que pertenezcan al conjunto del [0,9] de forma que satisfaga el $send + more = money$.

Figura 2: Problema de send more money.

Modelo:

- Definición de variables, una para cada letra: $s,e,n,d,m,o,r,y \in [0,9]$.
- Restricciones:
  - Cumplir con la regla de las numeración decimal:
    $10^3(s+m) + 10^2(e+o) + 10(n+r) + d + e = 10^4 m + 10^3 o + 10^2 + 10e + y$.
  - Todas las letras deben tener valores diferentes:
    $alldiferent(s,e,n,d,m,o,r,y)$

4.2 Algorítmo de Búsqueda

Los algoritmos de búsqueda son la base para la resolución de un CSP. Estos buscan en el espacio de estados del problema (posibles asignaciones de las variables) para encontrar una solución que satisfaga todas las restricciones del problema [15]. En el caso de no encontrar una solución estos deben probar que no existe.

El espacio que contiene el conjunto de variables puede ser representado como un árbol de búsqueda, en el cual cada nodo es una asignación parcial de valores a las variables, el nodo raíz del árbol de búsqueda representa el caso en el que ninguna variable se encuentra instanciada, y los nodos hojas son todas las variables que se encuentran instanciadas.

Las búsquedas pueden ser de dos tipos; completas si es que recorren todo el espacio y garantizan encontrar una solución e incompletas si utilizan algoritmos de búsqueda local y sólo recorren cierto espacio de soluciones. Estas últimas son más usadas ya que tienen un menor costo que las búsquedas completas.

Existen diversos algorítmos de búsqueda y cada uno utiliza un método diferente para encontrar una solución. Algunos pueden resultar mejores que otros en cuanto a su efectividad y/o rendimiento. Se explicarán brevemente los más comunes resaltando sus características, funcionamiento y principales desventajas.

4.2.1 Generate and Test (GT)

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. 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 debe instanciar 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, gran cantidad de asignaciones que podrían no llevar a la solución, por lo tanto es un algoritmo ineficiente para problemas con una gran cantidad de variables y con amplios dominios.

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 usar GT para este problema, ya que la gran mayoría de las primeras asignaciones (todas las variables han sido asignadas a un elemento del dominio) no satisfacen algunas de las restricciones y por lo tanto no son soluciones.

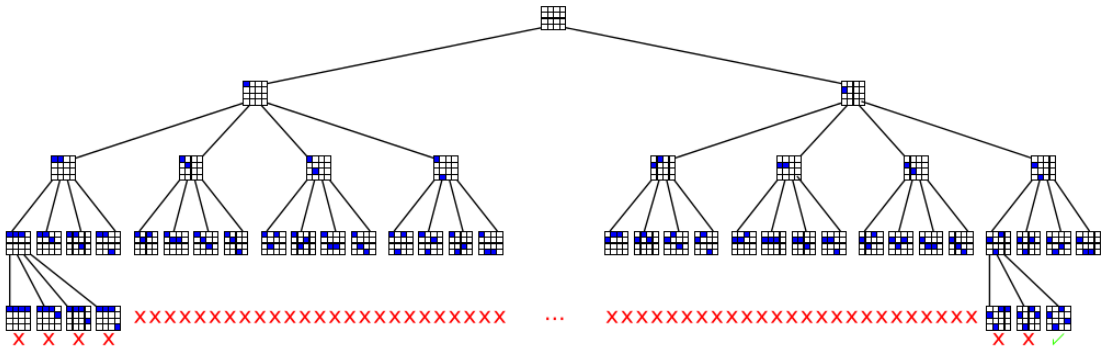

Figura 3: Generate and Test para el problema de 4-reinas.

4.2.2 Backtraking (BT)

También llamado Backtracking Cronológico, es una mejora al algoritmo Generate and Test. Esta mejora radica en que cada vez que se asigna un nuevo valor a una variable actual (R_i), se comprueba si este 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_i repitiendo el chequeo de consistencia, si se agotan todos los valores de su dominio, el algorítmo Backtracking retrocede para probar otro valor para la variable R_{i-1}, si tras asignarle otro valor y luego de comprobar 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.

En resumen la mejora que realiza es que si se detecta inconsistencia en la asignación parcial ésta se descarta, ya que en los siguientes subárboles no podrá existir 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.

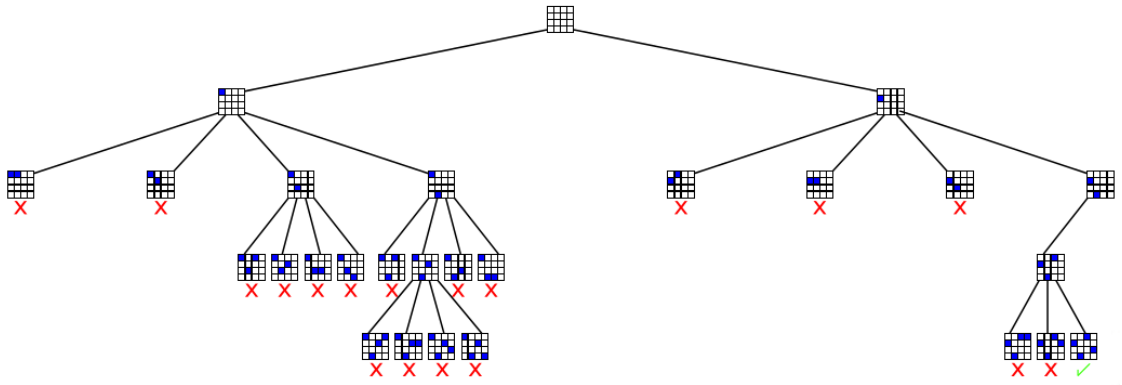

Figura 4: Backtraking para el problema de 4-reinas.

4.2.3 Forward Checking (FC)

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, si existen inconsistencias de la variable actual y las futuras entonces esos valores no serán instanciados, además realiza el chequeo de la variable actual con las pasadas.

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

Este algorítmo limita el espacio de búsqueda, ya que al revisar las variables futuras se reduce el posible dominio de las variables que serán instanciadas posteriormente. Pese a que es un algorítmo 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 algorítmo conoce los valores del dominio que pueden ser instanciados y posiblemente verificar la consistencia de éstos con los futuros.

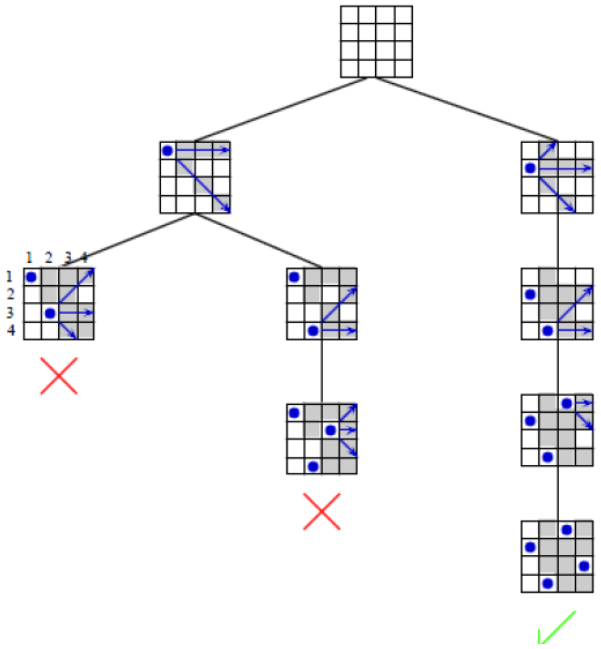

Figura 5: Forward Checking para el problema de 4-reinas.

4.2.4 Maintaining Arc Consistency (MAC)

MAC elimina inmediatamente de la primera asignación aquello que genere un conflicto posterior, luego con los elementos del dominio que estén aún disponibles realiza el mismo proceso verificando así inmediatamente si existe una solución en base a la primera asignación, o bien, devolviéndose y eliminando aquella asignación de no existir una solución con esas características.

La Figura 6 muestra el funcionamiento de este algorítmo 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) porque 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 ahora 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_4), por lo tanto para la posición de la primera reina en la celda (1,1) no existe solución. El algorítmo para otra primera reina en otra posición se repite de forma equivalente hasta

llegar a la solución como se muestra en la figura 6.

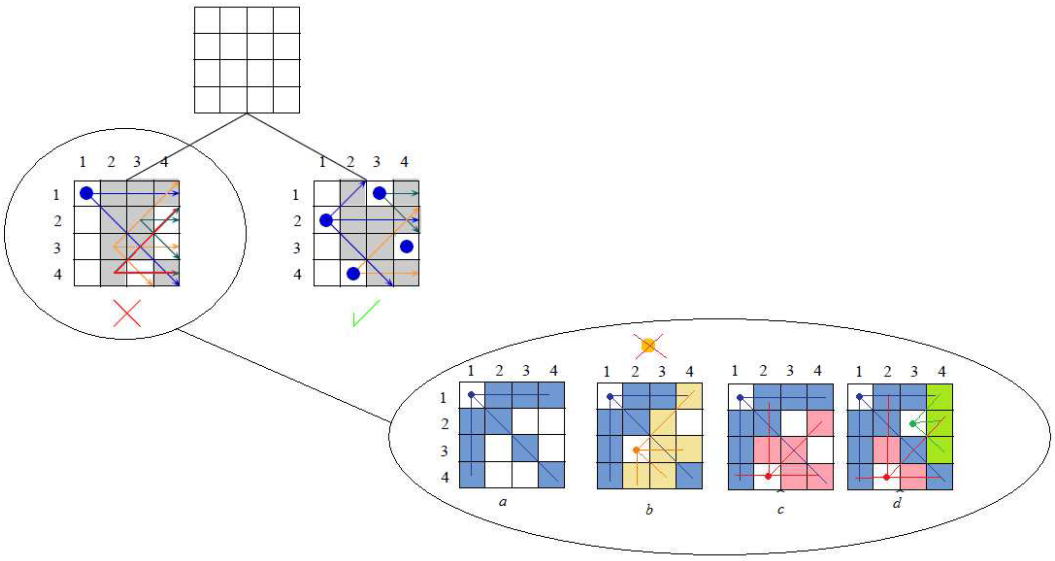

Figura 6: Maintaining Arc Consistency para el problema de 4-reinas.

4.3 Técnicas de Consistencia

Los algorítmos de búsqueda son un método bastante sencillo, sin embargo, frecuentemente sufren de una explosión combinatorial del espacio de búsqueda, y por lo tanto por si solos no son un método suficientemente eficiente para resolver un CSP. Una de las principales dificultades con las que se enfrentan estos algorítmos es la contínua aparición de inconsistencias locales. Las inconsistencias locales son valores individuales o combinación de valores de las variables que no pueden participar en la solución. Las técnicas de consistencia (o inferencia) pretenden reducir el espacio de búsqueda con un proceso de filtrado que elimina los valores inconsistentes de los dominios de las variables o inducen restricciones implícitas entre las variables, obteniendo así un nuevo CSP, igual al inicial, pero donde se han explicitado las restricciones que antes estaban implícitas. A continuación se explicarán las técnicas más usadas en los algorítmos de búsqueda:

4.3.1 Nodo-Consistencia

El funcionamiento de la nodo-consistencia se basa en asegurar que todos los valores en el dominio de una variable satisfagan las restricciones unarias sobre la variable. Un CSP es nodo-consistente si por cada valor del dominio de la variable x, cada restricción unaria sobre x es satisfecha. Si el dominio de la variable

x, contiene valores que no son satisfechos, entonces la inconsistencia del nodo puede ser eliminada, ya que no estaría contenida en ninguna solución. Este proceso se realiza eliminando los valores del dominio de X.

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

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

4.3.2 Arco-Consistencia

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

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

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

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

4.4 Estrategias de enumeración

Para el correcto funcionamiento de los algorítmos de búsqueda es necesario contar con el orden específico en que cada una de las variables será analizada, el orden en que los valores del dominio serán instanciados para cada variable y por último también se puede especificar el orden de las restricciones del problema. La correcta selección del orden para la variable y para el valor, puede mejorar significativamente la eficiencia de la solución generada. Para establecer estos órdenes es que existen las heurísticas de selección de variable y las heurísticas de selección de valor. Ambas heurísticas en conjunto forman las estrategias de enumeración. Cada una de estas distintas estrategias de enumeración tendrá un comportamiento distinto, por lo tanto, pueden existir algunas con un comportamiento más eficiente que las otras. De esta manera es que a continuación describiremos algunas heurísticas de selección de variable y de valor.

4.4.1 Heurísticas de selección de variable

La manera en la que se pueden ordenar las variables puede ser estática o dinámica. Estática si se asigna un orden fijo de las variables antes del inicio de la búsqueda. Además, cada nivel del árbol de búsqueda se asocia con una variable y también es necesario que cada nodo se asocie con una variable para la generación de sus sucesores. Dinámica si el orden de las variables puede cambiar basándose en información local que se genera durante el proceso de búsqueda. La heurística de selección de variable dinámica se representa en el árbol de búsqueda porque existen diferentes variables en un mismo nivel. Ambas heurísticas se basan en

la premisa de que es más conveniente asignar primero las variables que más restringen a las demás, de esta forma se identifican mucho antes las situaciones sin salida y por ende se reduce la cantidad de backtracks.

Las heurísticas de selección de variable que utilizamos 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. Utilizado normalmente si la variable representa un costo y estamos interesados en la elección que podría incrementar los costos si no se escoge a mejor alternativa. Desafortunadamente no siempre funciona: si dos variables de desición incurren en el mismo costo mínimo, el peso no se calcula como cero, pero como la diferencia de este valor mínimo al siguiente valor mayor.

4.4.2 Heurísticas de selección de valor

Requiere menos trabajo la ordenación de valores si se compara con la ordenación de variables. Básicamente, la idea de éstas heurísticas es la de seleccionar el valor de la variable actual que tenga más probabilidad de llegar a una solución, es decir, la rama del árbol de búsqueda donde sea más probable encontrar una solución. Estas heurísticas seleccionan, en su mayoría, el valor que restringe menos a la variable actual, o sea, el valor que menos reduce los valores útiles para las variables que aún no son instanciadas.

Las heurísticas de selección de valor que utilizamos son:

- **indomain:** Los valores son tratados en orden creciente. En caso de fallo, el valor previamente probado no se quita.
- **indomain_max:** Los valores son tratados en orden decreciente. En caso de fallo, se quita el valor previamente probado.
- **indomain_middle:** Los valores son tratados comenzando desde el centro del dominio. En caso de fallo, se quita el valor previamente probado.

5 Autonomous Search

Autonomous Search (AS) [17] es un caso particular de sistemas adaptativos que tiene como objetivo mejorar el rendimiento de la solución de un problema de satisfacción de restricciones, adaptándo su estratégia de búsqueda para el problema en cuestión. Posee la habilidad de modificar ventajosamente sus componentes internos cuando es expuesto a cambios externos y a oportunidades.

Su objetivo principal es mejorar el rendimiento de la solución. Esta mejora se puede medir a través del valor de ciertos indicadores o bien del tiempo que se demore en obtener la solución al problema, y esto lo logra adaptando la estratégia de búsqueda del problema. Los componentes internos que se pueden modificar, corresponden a varios algorítmos 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 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 estratégia a utilizar.

5.1 Arquitectura

Decidir qué heurísticas de selección de variable y de valor utilizar es un trabajo difícil. Aún cuando estas sean elegidas, no se asegura encontrar una solución para un problema de satisfacción de restricciones, ó si se encuentra la solución que ésta sea la óptima. En consecuencia es necesario llevar a cabo un proceso de resolución, donde será indispensable evaluar las estratégias, para encontrar la solución.

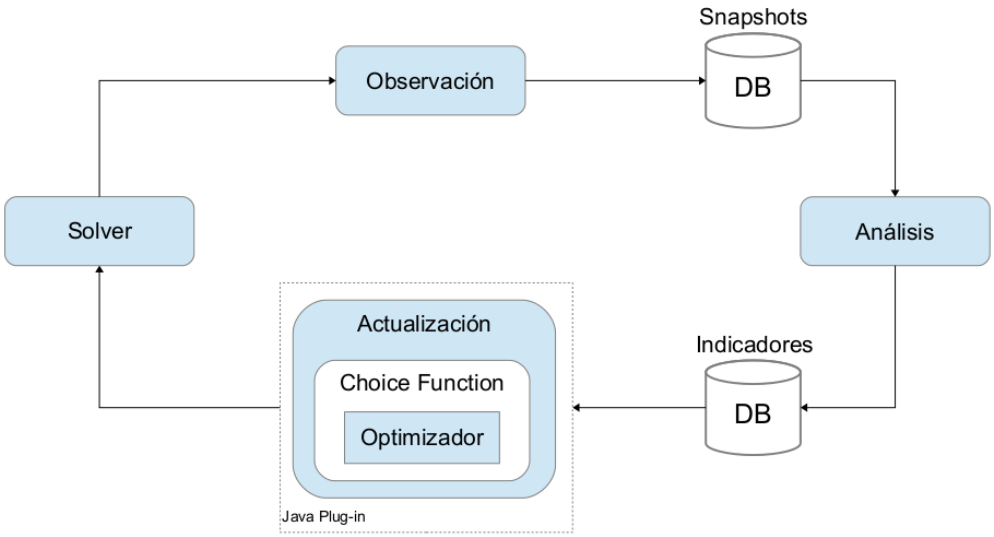

Figura 7: Esquema del Framework de Autonomous Search.

La arquitectura de AS se basada en 4 componentes (figura 7): solver, observación, análisis y actualización los cuales se describen a continuación:

- **Solver:** Componente que corre un algoritmo CSP genérico, alternando la propagación de restricciones y las fases de enumeración. Posee un conjunto de estratégias de enumeración básicas cada una caracterizada por una prioridad que evoluciona durante el cálculo, el componente actualización evalúa las estratégias y actualiza sus prioridades. Para cada enumeración (asignación un valor a una variable), la estratégia de enumeración dinámica selecciona la estratégia 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 realizados 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 estratégias 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 (*choice function*), ésta determina el desempeño (o rendimiento) de una estratégia dada en una determinada cantidad de tiempo. ésta puede ser calculada basada en los indicadores entregados por el componente

análisis.

5.2 Función de Selección

La tarea de la función de selección es hacer una comparación entre el rendimiento que tiene cada estratégia de enumeración históricamente y el punto de decisión que se está analizando en ese preciso momento. En cada punto de decisión es cuando se llama al solver para fijar una variable de enumeración.

Código	**Nombre**	**Descripción**
VFP	Variables fijadas por propagación en cada Paso	Número de variables reparadas por propagación.
Step	Número de mediciones	Cada vez que se usa la función de selección.
TV	Número total de variables	Número total de variables en el problema. Indicador en relación a las características del problema, este valor es constante durante el proceso.
VU	Variables no instanciadas	Número de variables no instanciadas.
SB	Shallow Backtrack	Cuando se trata de asignar un valor a la variable actual sin éxito entonces se recorre el próximo valor.
B	Backtracks	Cuando la variable actual lleva a una ruta sin salida, entonces se retrocede a la variable anterior.
N	Nodos	Cantidad de nodos visitados.
PTAVFES		Porcentaje total acumulado de variables reparadas por la enumeración en cada paso.
dmax	Profundidad máxima actual	La profundidad máxima actual en el árbol de búsqueda.
In1	Profundidad máxima actual - Profundidad máxima previa	Representa la variación de la profundidad.
In2	Profundidad actual - profundidad previa	Si es positivo significa que el nodo actual se encuentra a más profundidad que el del paso previo.
VF	Variables Instanciadas	Número de variables instanciadas por enumeración y propagación.
DB		Si se produce un backtrack, cuenta la cantidad de niveles hacia atrás (profundidad).

Tabla 1: Ejemplo de indicadores que pueden componer una función de selección.

En cada llamado, la función de selección rankea, califica y luego elige entre las estratégias de enumeración S_j, la función de selección f en el paso n por cada estratégia S_j está definida por 1, donde l es el número de indicadores considerados y α es un parámetro para controlar la relevancia del indicador dentro de la función de selección.

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

A esto, hay que agregar que para controlar la relevancia de un indicador i para una estratégia S_j en un intervalo de tiempo, se utiliza una técnica estadística para producir un suavizado exponencial. La idea de esto es darle mayor importancia a rendimientos más recientes, asociándoles pesos exponencialmente decrecientes 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 se define en las siguientes ecuaciones:

$$f_n(S_j) = v_0 \tag{2}$$

$$f_n(S_j) = v_n + \beta_i f_{in-1}(S_j) \tag{3}$$

Donde v_0 es el valor del indicador i para la estratégia S_j en la ecuación anterior, n es un paso dado del proceso, β es el factor de suavizado, y $0 < \beta < 1$.

De manera más gráfica, la labor de la función de selección es descrita en la figura 8

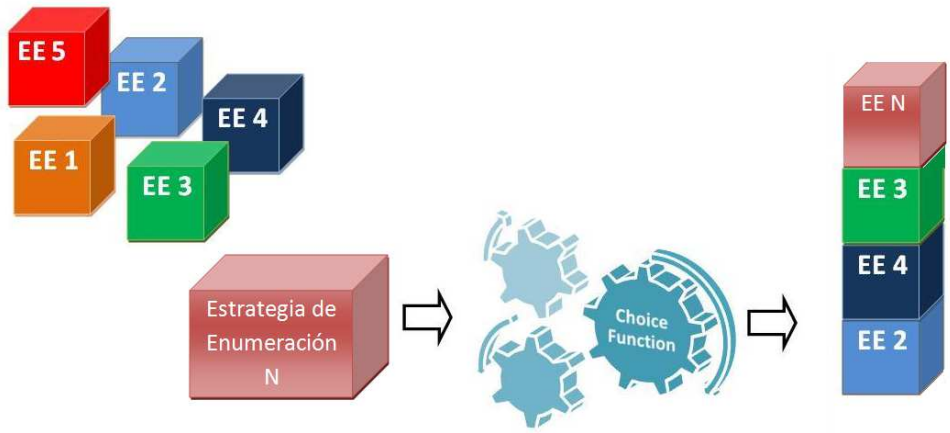

Figura 8: Ranking de las estratégias de enumeración.

5.2.1 Cálculo parámetro α

Cada indicador dentro de la función de selección tiene un grado de importancia que esta dado por cada α_i, el cual se utiliza en la ecuación 1. Dependiendo del indicador que estemos evaluando se debería

maximizar o minimizar el valor de α. De esta forma si se considera como un indicador la reducción del espacio de búsqueda, lo que se necesita es maximizar α. Es por esto que el cálculo de α puede ser planteado como un problema de optimización.

Este problema de optimización se pretende abordar implementando el optimizador GSA para que este pueda llevar un mejor control en linea de las estrategias de enumeración. GSA es un optimizador que se basa en las leyes Newtonianas de gravedad y movimiento, donde su rendimiento se mide a través de sus masas.

En el siguiente capítulo se explicará de manera más detallada su funcionamiento.

6 Gravitational Search Algorithm

El Gravitational Search Algorithm (GSA) es un algorítmo heurístico que se basa en las leyes Newtonianas de gravedad y movimiento [19]. En GSA, los agentes son considerados objetos y su rendimiento se mide a través de sus masas. Todos los agentes interactúan con cada uno de los demás agentes a través de las leyes de gravedad y movimiento de Newton [20][21]. Los objetos se atraen entre ellos a causa de la fuerza de gravedad, y esta fuerza produce un movimiento global de todos los objetos hacia aquellos objetos que poseen las masas mas pesadas. Estas masas corresponden a buenas soluciones, sin embargo, al ser más pesadas, se mueven mas lentas que aquellas masas pequeñas. Esto garantiza la explotación del algorítmo.

En GSA, cada agente (masa) posee cuatro especificaciones: su posición, su masa inercial, su masa gravitacional activa y su masa gravitacional pasiva. La posición corresponde a una solución para el problema, y su masa gravitacional e inercial se determinan utilizando una función de aptitud.

Cada agente (masa) representa una solución, y el algorítmo se recorre ajustando apropiadamente la masa gavitacional e inercial. Por intervalos de tiempo, esperamos que las masas sean atraídas por aquellas que poseen las masas más pesadas. Estas masas representarán soluciones óptimas en nuestro espacio de búsqueda.

En GSA, la posición i-ésimo agente se podría definir como:

$$X_i = (x_i^1, x_i^2, ..., x_i^d, ..., x_i^n) \qquad for \quad i = 1, 2, ..., N, \tag{4}$$

donde x_i^d es la posición del i-ésimo agente en la d-ésima dimensión, n es la dimensión del espacio de búsqueda, y N es el número de agentes. En un determinado tiempo 't', definiremos la fuerza que actúa sobre la masa 'i' desde la masa 'j' de la siguiente manera:

$$F_{ij}^d(t) = G(t)\frac{M_{pi}(t)M_{aj}(t)}{R_{ij}(t) + \varepsilon}(x_j^d(t) - x_i^d(t)) \tag{5}$$

donde M_{aj} es la masa gravitacional activa relacionada con el agente j, M_{pi} es la masa gravitacional pasiva relacionada con el agente i, $G(t)$ es la constante gravitacional en el tiempo t, ε es una pequeña constante, y $R_{ij}(t)$ es la distancia euclidiana entre el i-ésimo y el j-ésimo agente que se define de la siguiente manera:

$$R_{ij}(t) = \|X_i(t), X_j(t)\|_2 \tag{6}$$

En GSA, G es considerada una función linealmente decreciente de la siguiente manera:

$$G(t) = G_0(1 - \frac{t}{T}) \tag{7}$$

donde G_0 es una constante con valor inicial 100 y T es el numero total de iteraciones (la edad total del sistema). La fuerza total que actúa sobre el i-ésimo agente es de la siguiente manera:

$$F_i^d(t) = \sum_{j=1, j\neq i}^{N} rand_j F_{ij}^d(t), \tag{8}$$

suponemos que la fuerza total que actúa sobre el agente i en una dimesión d es una suma ponderada al azar de los d-ésimos componentes de las fuerzas ejercidas por los otros agentes. $rand_j$ es un número randómico en el intervalo [0, 1]. De acuerdo con la ley de movimiento de Newton, la aceleración del i-ésimo agente en el tiempo t, en la d-ésima dimensión, es de la siguiente manera:

$$a_i^d(t) = \frac{F_i^d(t)}{M_{ii}(t)} \tag{9}$$

donde M_{ii} es la masa inercial del i-ésimo agente. La siguiente velocidad de un agente se considera como una fracción de su actual velocidad más su aceleración. Por lo tanto, la posición y la velocidad del del i-ésimo agente en el tiempo t, en la d-ésima dimensión se podría formular de la siguiente manera:

$$v_i^d(t+1) = rand_i \times v_i^d(t) + a_i^d(t) \tag{10}$$

$$x_i^d(t+1) = x_i^d(t) + v_i^d(t+1) \tag{11}$$

donde $rand_i$ es una variable uniforme randómica en el intervalo [0, 1]. Usamos este numero randómico para darle una característica randómica a la búsqueda.

En GSA, las masas gravitacionales e inerciales son consideradas lo mismo. Sin embargo, estas pueden tener diferentes valores. Una gran masa inercial realiza una operación de búsqueda mas precisa debido al que el movimiento de los agentes es lento. Por otro lado, una gran masa gravitacional provoca una gran atracción, lo que permite una rápida convergencia. En consecuencia, actualizamos las masas gravitacionales e inerciales por la siguiente equación:

$$M_{ai} = M_{pi} = M_{ii} = M_i, \quad i = 1, 2, ..., N, \tag{12}$$

$$m_i(t) = \frac{fit_i(t) - worst(t)}{best(t) - worst(t)}, \tag{13}$$

$$M_i(t) = \frac{m_i(t)}{\sum_{j=1}^{N} m_j(t)} \tag{14}$$

donde $fit_i(t)$ representa el valor de aptitud del i-ésimo agente en el tiempo t, y, $worst(t)$ y $best(t)$ se definen de la siguiente manera:

Para problemas de minimización:

$$best(t) = \min_{j \in \{1,\ldots,N\}} fit_j(t) \tag{15}$$

$$worst(t) = \max_{j \in \{1,\ldots,N\}} fit_j(t) \tag{16}$$

Para problemas de maximización:

$$best(t) = \max_{j \in \{1,\ldots,N\}} fit_j(t) \tag{17}$$

$$worst(t) = \min_{j \in \{1,\ldots,N\}} fit_j(t) \tag{18}$$

Una forma de tener un buen compromiso entre la exploración y la explotación es reducir el número de agentes a medida que va pasando el tiempo (ecuación 8). De esta manera, proponemos que solo un subconjunto de agentes, aquellos que posean las masas mas grandes, apliquen su fuerza al resto. Sin embargo, hay que tener mucho cuidado al aplicar esta política, ya que esto podría reducir el poder de exploración e incrementar el poder de explotación.

Para evitar caer en un mínimo local, en un comienzo el algorítmo debe utilizar la exploración. Durante períodos de tiempo, la exploración debe desaparecer y la explotación debe aparecer. Para mejorar el rendimiento de GSA, sólo los $Kbest$ agentes atraerán a los demás. $Kbest$ es una función de tiempo, con K_0 al comienzo y decreciendo linealmente a través del tiempo. Por lo tanto, inicialmente, todos los agentes aplican su fuerza a los demás, y a medida que pasa el tiempo, $Kbest$ se reducirá linealmente provocando que al final solo un agente aplique su fuerza a todo el resto. De esta forma, la ecuación (8) pordría ser reemplazada por:

$$F_i^d(t) = \sum_{j \in Kbest, j \neq i} rand_j F_{ij}^d(t), \tag{19}$$

donde $Kbest$ es el subconjunto de los K agentes con el mejor valor de aptitud y las masas mas grandes. El algorítmo del GSA quedaría la siguiente manera (algorítmo 1):

Algoritmo 1 GSA

Cargar solución inicial a los agentes x_i.
Obtener fitness para cada agente x_i.
  while $it < it\_max$ **do**
    Actualizar $Kbest$ y G (7), además asignar $best$ y $worst$.
    for all x_i **do**
      Calcular m (13) y $Masa$ (12).
    end for
    for all x_i **do**
      for all $Kbest$ **do**
        if $x_i \neq Kbest$ **then**
          Calcular fuerza entre el x_i agente y el $Kbest$ agente (19).
        end if
      end for
      Para cada x_i calcular la aceleración (9), la nueva velocidad (10) y su nueva posición (11).
    end for
  end while
  return La mejor solución encontrada $best$.

Primeramente es necesario cargar la solución inicial a cada uno de los agentes y calcular su *fitness*. El *fitness* es un parámetro que se utilizará para determinar si un agente es mejor que otro. Si se trata de un problema de maximización se busca el agente que posee el mayor *fitness* como la mejor solución del sistema, y por el contrario, si se trata de un problema de minimización se buscará el menor *fitness* del sistema. En cada iteración debemos ordenar y actualizar los $Kbest$ agentes que ejercen su fuerza sobre todos los demás. Se actcualiza G y además se determina el mejor y peor agente del sistema ($best$ y $worts$). Para cada uno de los agentes es necesario el calculo de su masa (m y $Masa$). Con estos valores se calcula la fuerza ejercida por los $Kbest$ agentes sobre el resto del sistema. Finalmente se calcula la aceleración, la nueva velocidad y la nueva posición.

Se deben repetir estos pasos mientras no se cumpla el criterio de parada it_max. Una vez alcanzado este criterio se obtiene el mejor agente del sistema.

7 Experimentos

Los experimentos fueron realizados en equipos con las siguientes características:

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

Las características del equipo son:

- RAM: 4GB.
- Procesador: Intel Core i3 3.3GHz.
- Sistema Operativo: Windows 7 Professional de 32 bits.

En computadores con las características descritas se resolverán los siguientes problemas:

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

Las pruebas mencionadas anteriormente se realizaron con ECL^{i}PS^e Constraint logic Programming Solver v6.10, y el optimizador GSA ha sido desarrollado en el lenguaje de programación Java.

La combinación de las heurísticas de selección de variable y de valor generan veinticuatro estrategias de enumeración las cuales se muestran en la tabla 2.

Para los experimentos se utilizaron dos conjuntos de funciones de selección (*choice functions*). El primer conjunto es utilizado por los optimizadores (3), asignandoles diferentes grados de importancia a los indicadores en cada iteración dependiendo de su comportamiento en la resolución del problema. El segundo conjunto (4) no posee un optimizador, además de esto, fue removido el grado de importancia α para poder realizar pruebas y analizar su comportamiento.

Heurísticas de selección								
Id	**Variable**	**Valor**	Id	**Variable**	**Valor**	Id	**Variable**	**Valor**
S_1	**input_order**		S_9	**input_order**		S_{17}	**input_order**	
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**	**Mín**	S_{12}	**occurrence**	**Med**	S_{20}	**occurrence**	**Máx**
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 utilizadas en los experimentos.

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.

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.

Con el fin de poder encontrar la configuración correcta de GSA es que se realizó un sampling con el problema de las N-Queens (NQ=20). Los resultados obtenidos se muestran en la Tabla 5.

Iterations	Agents	*Runtime (ms)*
100	10	960,9
150	10	1392,4
100	20	1882,4
200	10	1883
100	30	2611,2
150	20	3006,1
200	20	3792,2
100	40	3856
150	30	4560,5
100	50	4830,4
150	40	5697,8
200	30	5740,6
150	50	7033,1
200	40	7549,2
200	50	9502,2

Tabla 5: Sampling para GSA.

Con la mejor configuración de GSA (100 iteraciones y 10 agentes) se probaron las diferentes funciones de selección para evaluar la que se comporta de mejor manera con la metaheurística, y de esta manera, poder realizar los experimentos. Luego de realizar el sampling, la función de selección que tuvo mejores resultados con la metaheurística es la CF_2. Los resultados obtenidos serán comparados con los optimizadores PSO y GA, y con un componente randómico RND que selecciona estrategias de enumeración al azar, los cuales que poseen resultados reportados en [22]. Cada instancia se resuelve un número máximo de 65,535 pasos. Si se llega al límite y no se ha encontrado solución, se considera como fallido y se asume un time-out (t.o.) como tiempo de resolución ó se antepone $>$ al número máximo de backtracks alcanzado.

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
GSA	1	0	0	1	2	0	2	0,85	0	2	37	292	0	66,2

Tabla 6: Backtracks obtenidos en las diferentes estrategias de enumeración en los problemas N-Queens y Sudoku.

En la Tabla 6 se presentan los backtracks obtenidos en la resolución de las diferentes instancias de N-Queens y del Sudoku respectivamente. Para el caso de las instancias del N-Queens podemos apreciar que de las 24 estrategias de enumeración sólo el 25% de éstas fue capaz de resolver todas las instancias, donde el promedio mínimo registrado fue de 148,14 backtracks, para el caso de las 12 funciones de selección que no poseen el parámetro α sólo el 16,6% fue capaz de resolver todas las instancias donde el menor numero promedio de backtracks registrado fue de 505,43. Por su parte los optimizadores PSO, GA, GSA y RND fueron capaces de resolver todas las instancias sin problemas, sin embargo, el que presento un menor promedio fue GSA con 0.85 backtracks. Las instancias del Sudoku tuvieron un comportamiento un poco diferente. De las 24 estrategias de enumeración el 50% fue capaz de resolver todas las instancias, la menor de ellas con un promedio de 231,8 backtracks. De las 12 funciones de selección sin el parámetro α

sólo el 16.6% fue capaz de resolver todas las instancias, promediando la menor de ellas 3341,4 backtracks. Para el caso de los optimizadores, el que tuvo un mejor comportamiento fue RND con un promedio de 47 backtracks, sin embargo, al tener un comportamiento randómico, no se puede asegurar alcanzar nuevamente este valor. GSA por su parte, si bien no fue superior que RND, si tuvo una menor cantidad de backtracks en comparación con las 24 estrategias de enumeración y las 12 funciones de selección sin α.

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
GSA	0	0	12	734	1874	524	0	0	0	0	0	0

Tabla 7: Backtracks obtenidos en las diferentes estrategias de enumeración, choice function sin α y optimizadores para los problemas Magic Square y Latin Square.

En la tabla 7 se presentan los resultados obtenidos al resolver las instancias del Magic Squares y Latin Square. Para el caso del Magic Square ninguna de las 24 estrategias de enumeración fue capaz de resolver todas las instancias, sólo el 50% de éstas logró resolver hasta la instancia 5MS. Lo mismo ocurrió con las 12 funciones de selección sin el parámetro α, sólo el 58,3% de éstas logró resolver hasta la instancia 5MS. Para el caso de los optimizadores sólo GSA fue capaz de resolver todas las instacias del Magic Square con

un promedio de 524 backtracks.

El caso de Latin Square fue completamente distinto, ya que tanto las funciones de selección como estrategias de enumeración fueron capaces de resolver en su mayoría todas las instancia del problema. Primeramente, un 75% de las estrategias de enumeración fue capaz de resolver todas las instancias, siendo la con menor promedio 0 backtracks. De igual manera, un 75% de las funciones de selección sin el parámetro fueron capaces de resolver todas las instancias del problema, donde el menor valor promedio obtenido es de 0 backtracks. Finalmente, todos los optimizadores resuelven todas y cada una de las instancias con un backtrack igual a 0.

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
GSA	1	3284	1642,5	0	0	0	0	0	0	0	0	0	0	0	0

Tabla 8: Backtracks obtenidos en las diferentes estrategias de enumeración, choice function sin α y optimizadores para los problemas Knight's Tour, Quasigroup y Langford.

La tabla 8 contiene los resultados obtenidos en las diferentes instancias de los problemas Knight's Tour, Quasigroup y Langford. Para las instancias 2 instancias del Knight's Tour sólo el 25% de las estrategias de enumeración fue capaz de resolverlas, donde el mínimo valor promedio obtenido fue de 19231 backtracks. Ningúna función de selección fue capaz de resolver todas las instancias, y por su parte, los optimizadores no tuvieron problemas en la resolución, siendo el menor promedio obtenido 1642,5 backtracks correspondientes a GSA. Para las 5 instancias del Quasigroup se observa lo siguiente; el 33.33% de las estrategias de enumeración es capaz de resolver todas las instancias, donde se observa que 18 es el promedio mas bajo de backtracks obtenido; el 16.6% de las funciones de selección resolvió todas las instancias, donde el promedio mas bajo registrado corresponde a 82,2 backtracks; para el caso de los optimizadores, sólo GA tuvo backtracks en 2 de las 5 instancias, promediando 1554,3 backtracks, el resto resolvió todas las instancias con 0 backtracks. Los experimentos a las 5 instancias del Langford presentaron los siguientes resultados; el 87,5% de las estrategias de enumeración fue capaz de resolver todas las instancias, registrando un valor promedio mínimo de 16,6 backtracks; el 75% de las funciones de selección fue capaz de resolver todas las instancias, donde el valor promedio mínimo registrado fue de 65 baktracks; finalmente, los optimizadores resolvieron en su totalidad las instancias del problema, sin embargo sólo RND y GSA lo lograron con un promedio de 0 backtracks.

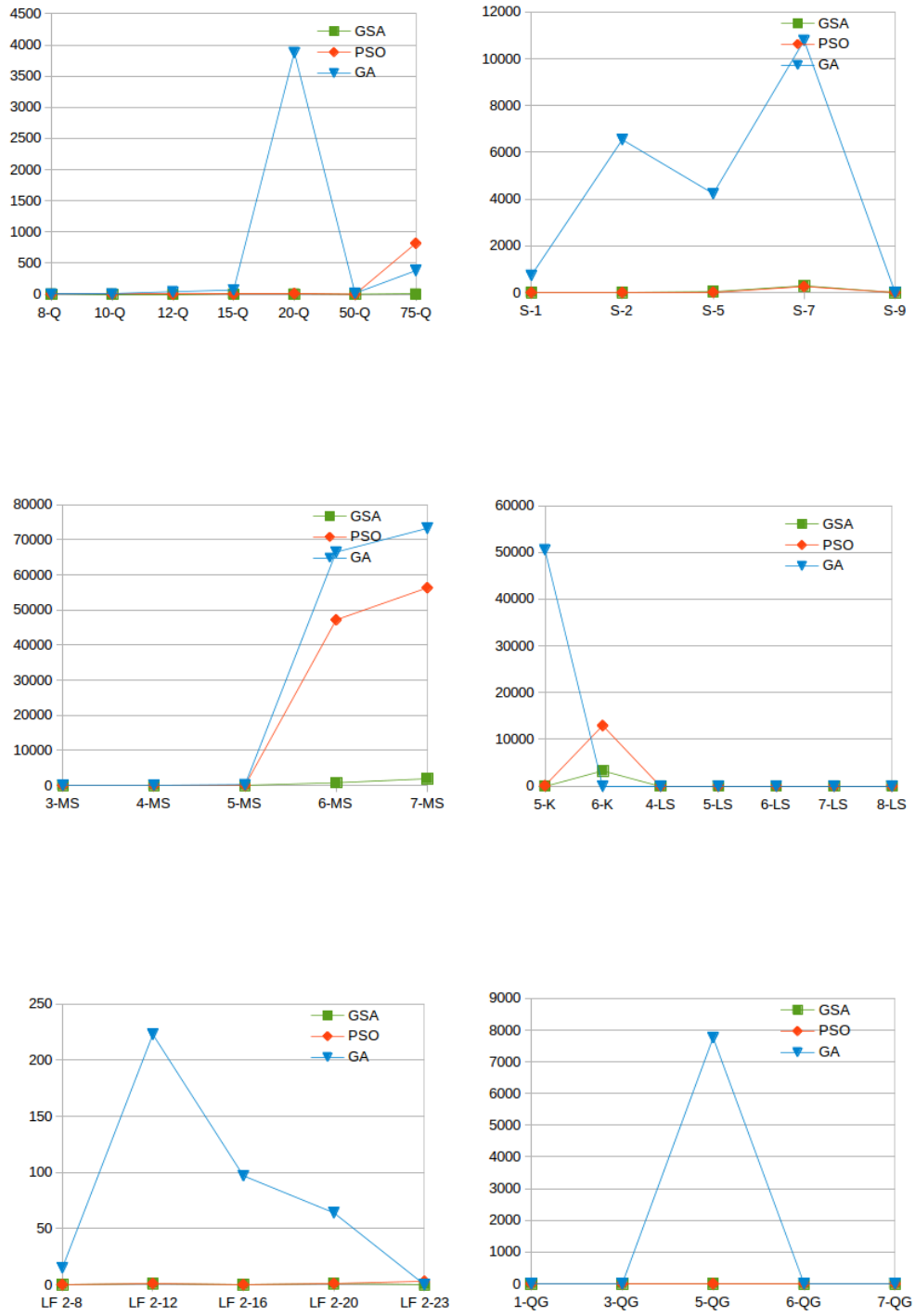

Figura 9: Backtracks observado en los diferentes optimizadores

En la figura 9 se puede observar de manera mas gráfica el comportamiento en la resolución de las instancias de todos los problemas anteriormente descritos. Si bien en algunas instacias GSA no obtuvo el mejor resultado, si fué el único capaz de resolver todos los problemas sin alcanzar el criterio de parada.

A continuación analizaremos los tiempos de resolución registrados para todos los problemas en cada una de sus instancias:

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
GSA	625	779	972	1322	2046	10760	28254	6394	879	760	836	847	841	832,6

Tabla 9: Solving Time obtenido en las diferentes estrategias de enumeración en los problemas N-Queens y Sudoku.

Para las instancias del N-Queens el 25% de las estrategias de enumeración logró resolver todas las instancias, de estos, el que lo hizo en el menor tiempo promedio fue en 4187 milisegundos; del 16,6% de las funciones de selección sin el parámetro α que que logró resolver todas las instancias, el que lo hizo en el menor tiempo promedio fue en 724,43 milisegundos. Para el caso de los optimizadores el que pudo resolver todas las instancias en el menor tiempo promedio es RND en 23,43 milisegundos, debido a que selecciona

randómicamente las estrategias de enumeración a utilizar en cada iteración. Al ser randómica no asegura que la siguiente ejecución sea igual de rá pida o bien, que encuentre la solución a todas las instancias de un determinado problema.

Para el Sudoku, el 50% de la estrategias de enumeración fue capaz de reolver todas las instancias, donde la que resolvió en el menor tiempo promedio lo hizo en 743,25 milisegundos; por su parte el 25% de las funciones de selección que pudieron resolver todas las instancias, el menor tiempo promedio registrado fue de 516,25 milisegundos; en los optimizadores no hubo problemas en la resolución, donde nuevamente el que resolvió las instancias en el menor tiempo promedio fue RND en 187,5 milisegundos seguido de GSA en 832,6.

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
GSA	731	1053	1004	1526	2654	1393,6	425	482	556	610	733	561,2

Tabla 10: Solving Time obtenidos en las diferentes estrategias de enumeración en los problemas Magic Square y Latin Square.

Ninguna estrategia de enumeración fue capaz de resolver todas las instancias del Magic Square, al igual que, ningúna función de selección sin el parámetro α. Los optimizadores por su parte también presentaron inconvenientes en la resolución de las instancias del problema ya que tanto RND, PSO y GA no fueron capaces de resolver las 2 últimas (6M y 7M). El único optimizador que pudo resolver todas estas instancias fue GSA, el cual promedió un tiempo de 1393,6 milisegundos.

Para las 5 instancias del Latin Square se observa lo siguiente; el 75% de las estraťegias de enumeración no tuvo problemas en la resolución, y de esta manera, el mínimo tiempo promedio que se registró fue de 5,2 milisegundos. El 83,3% de las funciones de selección sin el parámetro α tuvo buenos resultados en la resolución, obteniendo un mínimo tiempo promedio de 109,2 milisegundos. Los optimizadores por su parte, nuevamente fueron todos capaces de obtener solución a todas y cada una de las instancias del problema, siendo el que obtuvo un menor tiempo promedio RND con 7 milisegundos, seguido de GSA con 561,2.

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
GSA	4383	6508	5445,5	295	254	824	845	777	599	556	679	829	942	1054	812

Tabla 11: Solving Time obtenidos en las diferentes estrategias de enumeración en los problemas Knight's Tour, Quasigroup y Langford.

Para las 2 instancias de Knight's Tour tenemos los siguientes resultados; sólo el 25% de las estrategias de enumeración pudo resolver ambas instancias sin inconveniente en un tiempo acotado, registrando de esta manera un tiempo promedio mínimo de 1705 milisegundos. Para las funciones de selección sin el parámetro α, lamentablemente no se obtuvieron buenos resultados, ya que ningúna fue capaz de resolver en un tiempo acotado alguna de las instancias, sin embargo, los optimizadores pudieron resolver en su totalidad el problema, donde el tiempo promedio mínimo registrado fue de 5445,5 milisegundos, el cual corresponde a GSA.

Para las 5 instancias del Quasigroup se obtuvieron los siguientes resultados; sólo el 33,3% de las estrategias de enumeración pudo resolver todas las instancias del problema, registrando un tiempo promedio mínimo de 52,33 milisegundos. Por su parte, sólo un 16,6% de las funciones de selección sin el parámetro α fueron capaces de resolver todas las instancias del problema, registrando de esta manera un mínimo tiempo promedio de 357,67 milisegundos. Los optimizadores en su totalidad fueron capaces de resolver el problema, siendo RND el que lo hizo en el menor tiempo promedio con 15,67 milisegundos, seguido nuevamente por GSA con 599 milisegundos.

Para las 5 instancias del Langford se obtuvieron los siguientes resultados; el 87,5% de las estrategias de enumeración fue capaz de resolver las instancias del problema, registrando un tiempo promedio mínimo de 57,4 milisegundos. El 75 % de las funciones de selección sin el parámetro α pudieron resolver el problema en su totalidad, registrando un tiempo promedio mínimo de 244,4 milisegundos. Una vez mas todos los optimizadores pudieron resolver todas y cada una de las instancias del problema, siendo el menor tiempo promedio de resolución 20 milisegundos, que corresponde a RND, seguido de 812 milisegundos correspondientes a GSA.

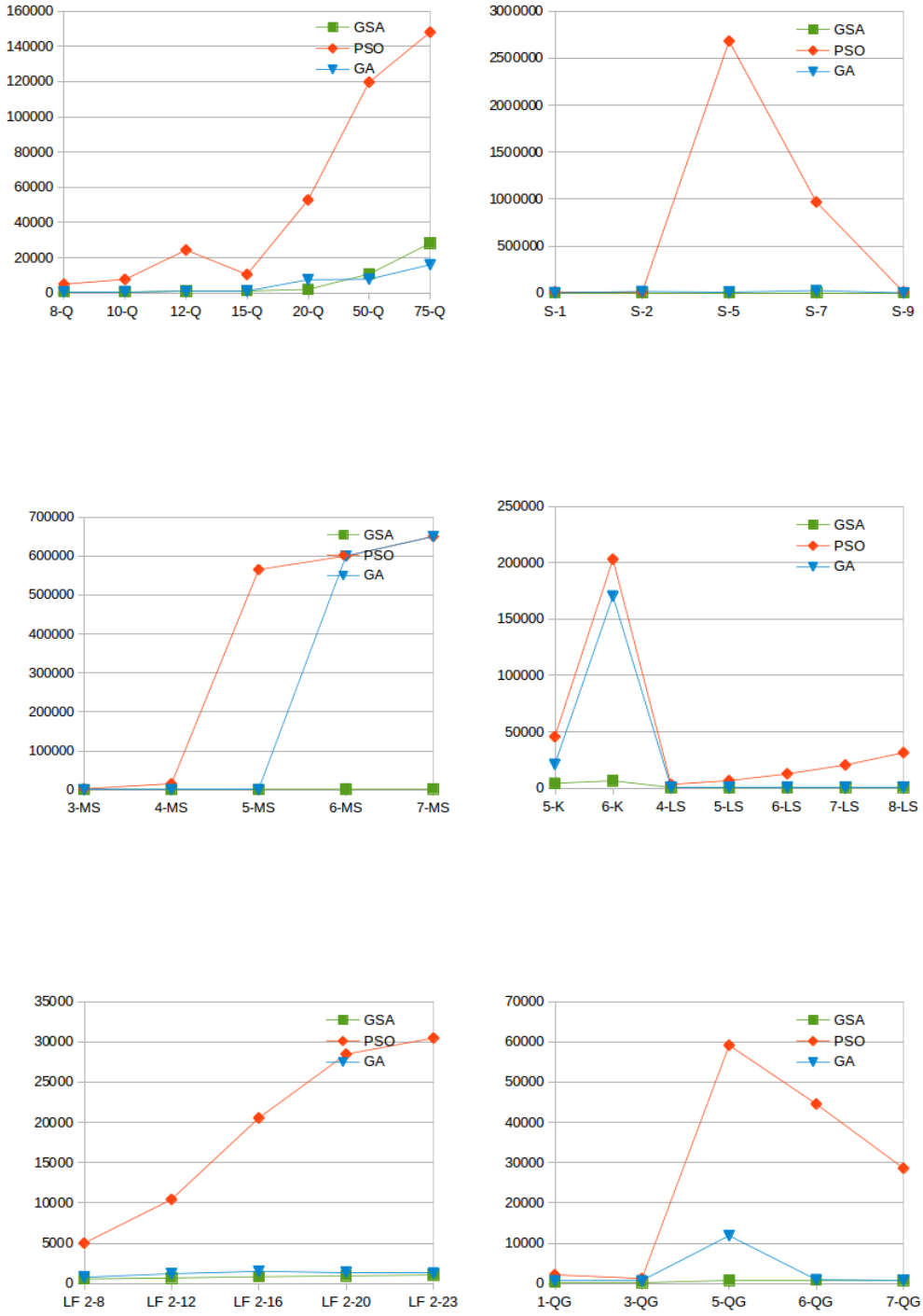

Figura 10: Solving time observado en los diferentes optimizadores

En la figura 10 se puede observar de manera gráfica la variación de los tiempos de resolución de los optimizadores frente a las diferentes instancias de los problemas anteriormente descritos.

GSA es un optimizador que, debido a su complejidad, no puede en muchas ocaciones ser mas rápido que los demás optimizadores con los que se realizaron las comparaciones y/o incluso con algunas estrategias de enumeración, sin embargo, fue el único capaz de resolver las instancias 6MS y 7MS del Magic Square.

8 Conclusión

La presente investigación tiene como objetivo mejorar el desempeño de la función de selección incorporando un nuevo optimizador al componente de actualización del framework de Autonomous Search.

Sin embargo, para llegar a esto se tuvieron que tocar muchos tópicos como pogramacion con restricciones, problemas de satisfaccion de restricciones, heurísticas de selección de valor y variable, wsm tunning entre otros. Se incorpó un nuevo optimizador, el Gravitational Search Algorithm, y con esto se generaron nuevos experimentos con el fin de poder encontrar mejores resultados en un menor tiempo de resolución.

Gratamente se aprecia que el optimizador GSA arroja buenos resultados en todas las instancias evaluadas para cada uno de los problemas. Si bien no fueron los mejores resultados en cuanto a tiempo o número de backtracks en comparación con los otros optimizadores (PSO y GA) fue el único capaz de resolver todos los problemas.

Todo esto en conjunto abre la puerta a posibles trabajos futuros, ya sea modificando la forma en que el solver realiza la propagación, incorporando nuevas estrategias de enumeración, nuevos optimizadores, etc.

A Anexos

A.1 Algoritmos Prolog

A.1.1 N-Queens

El algoritmo 2, implementa el modelo asociado al problema de las N-Reinas, desarrollado en el lenguaje de programación lógica **Prolog**.

Algoritmo 2 Algoritmo para el prolema N-Reinas en ECL^{i}PS^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.1.2 Sudoku puzzle

El algoritmo 3, implementa el modelo asociado al problema Sudoku Puzzle, desarrollado en el lenguaje de programación lógica **Prolog**.

Algoritmo 3 Algoritmo para el prolema Sudoku en ECL^{i}PS^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.1.3 Magic Squares

El algoritmo 4, implementa el modelo asociado al problema Magic Squares, desarrollado en el lenguaje de programación lógica **Prolog**.

Algoritmo 4 Algoritmo para el prolema Magic Squares en ECL^{i}PS^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.1.4 Knight's Tour

El algoritmo 5, implementa el modelo asociado al problema Knight's Tour, desarrollado en el lenguaje de programación lógica **Prolog**.

Algoritmo 5 Algoritmo para el prolema Knight Tour en ECL^{i}PS^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.1.5 Latin Squares

El algoritmo 6, implementa el modelo asociado al problema Latin Square, desarrollado en el lenguaje de programación lógica **Prolog**.

Algoritmo 6 Algoritmo para el problema Latin Square en ECL^{i}PS^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.1.6 Langford

El algoritmo 7, implementa el modelo asociado al problema Langford, desarrollado en el lenguaje de programación lógica **Prolog**.

Algoritmo 7 Algoritmo para el problema Langford en ECL^{i}PS^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.1.7 Quasigroup

El algoritmo 8, implementa el modelo asociado al problema Quasigroup, desarrollado en el lenguaje de programación lógica **Prolog**.

Algoritmo 8 Algoritmo Quasi Group en ECL^{i}PS^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] B. Crawford, C. Castro, E. Monfroy. *Programación con restricciones dinámicas.* 2009.

[2] F. Rossi, P. van Beek, T. Walsh. *Handbook of Constraint Programming.* 2006.

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

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

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

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

[7] B. Crawford, C. Castro, E. Mofroy, R. Soto, W. Palma and F. Paredes. *Dynamic Selection of Enumeration Strategies for Solving Constraint Satisfaction Problems.* Romanian Journal of Information Science And Technology Vol.15, pp. 106-128, 2012

[8] R. Soto, H. Kjellerstrand, J. Gutierrez, A. Lopez, B. Crawford, and E. 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.

[9] B. Crawford, C. Castro, E. Monfroy, R. Soto, W. Palma, and F. 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.

[10] R. Pizarro, G. 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.

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

[12] Y. Hamadi, E. Monfroy, F. Saubion. *What is Autonomous Search?.* Springer, 2008.

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

[14] K. Galleguillos. *Choice Function basada en Skyline para Autonomous Search.* Informe final de proyecto para optar al título de ingeniero de ejecución informática, Pontificia Universidad Católica de Valparaíso, 2013.

[15] I. Rodríguez. *Algoritmos de planificación basados en restricciones para la sustitución de componentes defectuosos.* Informe final para grado de Ph.D. en Ingeniería Informática, Universidad de Sevilla, 2008.

[16] R. Krzysztof, G. Wallace. *Constraint Logic Programming using Eclipse.* 2006.

[17] Y. Hamadi, E. Monfroy, F. Saubion. *What is Autonomous Search.* 2008.

[18] B. Crawford, R. Soto, M. Montecinos, C. Castro, and E. 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.

[19] E. Rashedi, H. Nezamabadi-pour, S. Saryazdi *GSA: A Gravitational Search Algorithm*, Department of Electrical Engineering, Shahid Bahonar University of Kerman, P.O. Box 76169-133, Kerman, Iran. 2009.

[20] D. Holliday, R. Resnick, J. Walker. *Fundamentals of physics.*, Fundamentals of physics. John Wiley and Sons;1993.

[21] B. Schutz. *Gravity from the ground up.*, Gravity from the ground up. Cambridge University Press; 2003.

[22] B. Crawford, R. Soto,E. Monfroy, W. Palma, C. Castro, F. Paredes. *Parameter tuning of a choice-function based hyperheuristic using Particle Swarm Optimization.* 2012.

[23] Maturana, J., Lardeux, F., Saubion, F., 2010. *Autonomous operator management for evolutionary algorithms.* J. Heuristics 16 (6), 881-909.

[24] Maturana, J., Saubion, F., 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.

[25] J. Maturana, F. Saubion, 2007. *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.* Springer, pp. 303-315.

[26] J. Maturana, A. Fialho, F. Saubion, M. Schoenauer, F. Lardeux, M. S. *Autonomous Search. Springer, Ch. Adaptive Operator Selection and Management in Evolutionary Algorithms.* 2012.

[27] T. StÃijtzle, M. L.-I., P. Pellegrini, M. Maur, M. de Oca, M. Birattari, M. Dorigo. *Autonomous Search. Springer, Ch. Parameter Adaptation in Ant Colony Optimization.* 2012.

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

[29] F. Hutter D Babic, H. Hoos, A. 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.

[30] F. Hutter, Y. Hamadi, H. Hoos, K. Leyton-Brown. *Performance prediction and automated tuning of randomized and parametric algorithms. In: CP. pp. 213-228.* 2006.

[31] L. Xu, F. Hutter, H. Hoos, K. Leyton-Brown. *Satzilla: Portfolio-based algorithm selection for sat. J. Artif. Intell. Res. (JAIR) 32, 565-606.* 2008.

[32] Y. Malitsky, M. Sellmann. *Instance-specific algorithm configura- tion 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.* 2012.

[33] F. Hutter, H. Hoos, K. Leyton-Brown. *Sequential model-based optimization for general algorithm configuration. In: In Proceedings of 28the 5th International Conference on Learning and Intelligent Optimization (LION). Vol. 6683 of LNCS. Springer, pp. 507-523.* 2011.

[34] F. Hutter, H. Hoos, K. Leyton-Brown. *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.* 2010.

[35] S. Epstein, E. Freuder, R. Wallace, A. Morozov, B. Samuels. *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.* 2002.

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

[37] Y. Xu, D. Stern, H. Samulowitz. *Learning adaptation to solve constraint satisfaction problems. In: Proceedings of the 3rd International Conference on Learning and Intelligent Optimization (LION). pp. 507-523.* 2009.

[38] F. Boussemart, F. Hemery, C. Lecoutre, L. Sais. *Boosting systematic search by weighting constraints. In: Proceedings of the 16th Eureopean Conference on Artificial Intelligence (ECAI). IOS Press, pp. 146-150.* 2004.

[39] D. Grimes, R. Wallace. *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.* 2007.

[40] R. Wallace, D. Grimes. *Experimental studies of variable selection strategies based on constraint weights. J. Algorithms 63 (1-3), 114-129.* 2008.

[41] R. Barták. *Constraint Programming: In Pursuit of the Holy Grail. Charles University, Faculty of Mathematics and Physics, Department of Theoretical Computer Science Malostranské námustí 2/25, 118 00 Praha 1, Czech Republic* 1999.