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

SKYLINE PARA AUTONOMOUS SEARCH

GIOVANNI ANTONIO AGUILERA QUIROZ
CRISTIAN EUGENIO SILVA BARRERA

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

JUNIO 2014

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

SKYLINE PARA AUTONOMOUS SEARCH

GIOVANNI ANTONIO AGUILERA QUIROZ
CRISTIAN EUGENIO SILVA BARRERA

Profesor Guía: **Wenceslao Palma Muñoz**
Profesor Correferente: **Ricardo Soto de Giorgis**

Carrera: **Ingeniería de Ejecución en Informática**

JUNIO 2014

Dedicatoria

Para todos los que nos apoyaron en el proceso. Especialmente a nuestros familiares y amigos.

Índice

Índice de Ilustraciones

Índice de Figuras

Índice de Tablas

Resumen

Resolver un Problema de Satisfacción de Restricciones (CSP) por medio de la Programación con Restricciones (CP) implica la exploración de un árbol de búsqueda donde las potenciales soluciones están distribuidas. La fase de exploración es controlada esencialmente por una estrategia de enumeración que decide el orden en el cual las variables y valores son seleccionados al momento de buscar una solución. Este proceso es muy importante ya que ciertas enumeraciones pueden llegar a una solución sin exploraciones inútiles. Sin embargo, seleccionar buenas estrategias con antelación es muy difícil ya que los efectos a lo largo de la búsqueda son a menudo impredecibles.

Autonomous Search (AS) aborda este problema proponiendo reemplazar en tiempo de ejecución estrategias con mal rendimiento por las más promisorias. Las estrategias son seleccionadas a partir de un ranking que se genera en función de su rendimiento en el proceso de resolución. Sin embargo, el ranking de estrategias de enumeración se apoya en un optimizador lo cual involucra un proceso computacional costoso. En el presente trabajo se propone un enfoque basado en una técnica de bases de datos llamada Skyline que evita el uso de funciones de ranking y optimizadores.

Palabras Claves: Autonomous Search, Problema de Satisfacción de Restricciones, Programación con Restricciones, Skyline.

Abstract

Solving a Constraint Satisfaction Problem (CSP), via a Constraint Programming (CP), approach, involves the exploration of a search tree where the potential solutions are distributed. The exploration phase is essentially controlled by an enumeration strategy that decides the order in which variables and values are selected to verify its feasibility. This process is known to be quite important, indeed perfect enumerations can reach a solution without useless explorations. However, selecting good strategies in advance is quite hard as the effect along the search are often unpredictable. Autonomous Search (AS) addresses this concern proposing to replace on the fly time bad-performing strategies by more promising ones. Strategies are selected from a quality rank which is generated in function of their performance on the current solving process. However, the ranking of the enumeration strategies is supported by an optimizer which is a costly process that tunes the computation of the ranking function. In this work, we propose an approach, in order to boost the solving process, based on the powerful database technique called Skyline which avoids the use of the rank functions and optimizers.

Keywords: Autonomous Search, Constraint Satisfaction Problem, Constraint Programming, Skyline.

1 Introducción

La Programación con Restricciones (CP) es una poderosa tecnología de software utilizada en la solución eficiente de problemas basados en restricciones. CP rescata ideas desde diferentes dominios tales como investigación de operaciones, inteligencia artificial, teoría de grafos y lenguajes de programación para resolver problemas en el contexto de computación gráfica, procesamiento del lenguaje natural, sistemas de bases de datos y biología molecular. El principio de CP es simple: el usuario formula un problema y el sistema lo resuelve.

En CP un problema es formulado como un Problema de Satisfacción de Restricciones (CSP) que consiste en una representación formal expresada en términos como una secuencia de variables y un conjunto de restricciones. El objetivo es encontrar valores para cada una de las variables y al mismo tiempo satisfacer las restricciones que las involucran. Para resolver un CSP se construye un árbol, que contiene potenciales soluciones, mediante el uso de backtracking. En general, dos grandes fases están presentes en el proceso de encontrar una solución: enumeración y propagación. La fase de enumeración otorga valor a las variables con el objetivo de crear ramas del árbol. La fase de propagación trata de podar el árbol filtrando desde el dominio de las variables aquellos valores que no conducen a una solución. En la fase de enumeración existen dos decisiones importantes que tomar: el orden y el valor en el cual las variables y sus valores son escogidos. Dicha selección se involucra heurísticas de selección de variable y heurísticas de selección de valor, y en conjunto se conocen como estrategia de enumeración.

La estrategia de enumeración es crucial en el rendimiento del proceso de solución de un CSP ya que una correcta selección puede reducir considerablemente el costo computacional de encontrar una solución. Sin embargo, decidir a priori una buena heurística es muy difícil ya que los efectos de la estrategia sobre el proceso de solución son impredecibles. Autonomous Search (AS) aborda dicha situación proponiendo el reemplazo de estrategias de mal rendimiento por otras de buen rendimiento durante todo el proceso de solución.

En los trabajos actuales el reemplazo de una estrategia depende de un ranking calculado usando una función de selección la cual a su vez considera varios indicadores presentes en el proceso de solución. Dicho ranking es calculado usando una hyperheurística donde los parámetros de la función de selección son afinados mediante el uso de un algoritmo genético lo cual ralentiza el proceso de solución de un CSP.

En el presente trabajo se propone un enfoque que evita el uso de una función de ranking y de un optimizador. La técnica utilizada se denomina Skyline la cual se utiliza en el dominio de las bases de datos para responder consultas basadas en múltiples criterios. Para evaluar el enfoque propuesto se realizarán pruebas que evaluarán su rendimiento al resolver problemas tales como Magic Square, Knight Tour, N-Queens y Sudoku.

2 Objetivos

2.1 Objetivo General

Implementar un algoritmo Top K que realice un ranking sobre los puntos Skyline (estrategias de enumeración) para así encontrar la mejor estrategia.

2.2 Objetivos Específicos

- Investigar en qué consiste la arquitectura de Autonomous Search.
- Estudiar algoritmos de Skyline.
- Comprender el algoritmo basado en Top K para determinar qué punto del conjunto Skyline es el mejor.
- Realizar pruebas con problemas como Magic Square, Knight Tour, N-Reinas y Sudoku.
- Realizar pruebas comparativas entre algoritmo de Skyline con Trade Off, Skyline sin Top-K y el algoritmo Skyline con Top-K que se está desarrollando.

3 Programación con restricciones

La programación con restricciones es un paradigma utilizado para la solución de problemas combinatoriales difíciles presentes en diversos dominios de la ciencia de la computación. La idea básica de la programación con restricciones es considerar que el usuario plantea una serie de restricciones y un solver es utilizado para satisfacerlas. El inicio de este paradigma se registra especialmente en la década de los 60' en el área de la inteligencia artificial para el entendimiento de lenguaje natural, razonamiento temporal y espacial, probador de teoremas y razonamiento cualitativo, tanto en la robótica como en los agentes y en la Bioinformática. En el año de 1963 se desarrolló Sketchpad que fue utilizado para sistemas gráficos para dibujos geométricos. Algunas de las áreas que estudia la aplicación de este paradigma son los siguientes [3]:

- Diseño y construcción de circuitos integrados: localización de fallas en los circuitos, pruebas y verificación del diseño.
- Biología molecular: secuencia de ADN, construcción de modelos 3D de las proteínas.
- Aplicaciones de negocios (comercio).
- Álgebra computacional: resolución y/o simplificación de ecuaciones sobre varias estructuras algebraicas.
- Problemas de optimización.

Los lenguajes de programación con restricciones son habitualmente ampliaciones de otro lenguaje. El primer lenguaje utilizado para tal efecto fue Prolog. Por esta razón es que este campo fue denominado inicialmente Programación Lógica con Restricciones. Ambos paradigmas comparten características muy parecidas, tales como las variables lógicas (una vez que una variable es asignada a un valor, no puede ser cambiado), o el backtracking [6].

La programación con restricciones se puede utilizar tanto para resolver problemas de optimización (COP, Constraint Optimization Problem) tan solo agregándole una función objetivo como maximizar o minimizar, como también para los Problemas de Satisfacción de Restricciones (CSP, Constraint Satisfaction Problem) los cuales consisten en una serie de variables limitadas o finitas contando cada una de estas con su dominio y restricciones, dado esto el CSP logra encontrar valores a cada variable descrita satisfaciendo todas las restricciones al mismo tiempo.

Existen distintos problemas que se pueden resolver con la programación con restricciones los cuales varían según su grado de complejidad como podría ser un simple coloreado de mapa que consta de pocas restricciones o como también un sistema de planificación de líneas aéreas.

Para la programación con restricciones existen algoritmos de búsqueda que sirven para dar solución a los problemas de satisfacción de restricciones (CSP). Ésta búsqueda considera todas las restricciones mientras le asigna valores a las variables en un espacio definido, dando solución al problema o justificando que ésta no exista. Algunos ejemplos de algoritmos de búsqueda son arc consistency, generate and test, forward checking y backtracking.

3.1 Problema de Satisfacción de Restricciones

Muchos de los problemas que se plantean en la Inteligencia Artificial pueden formalizarse como problemas de satisfacción de restricciones (CSP).

De esta manera, podemos definir formalmente a un problema de satisfacción de restricciones (CSP) como un 3-tupla $P = \{X, D, C\}$ donde:

- Ej. un conjunto finito de variables $X = \{X_1,..., X_n\}$.
- Ej. un conjunto finito de Dominios D_i asociados a cada variable X_i, especificando los posibles valores que puede tomar.
- Ej. un conjunto finito de restricciones $C_1,..., C_m$ que definen una serie de propiedades que deben verificar los valores asignados a las variables.

Una de las soluciones al problema es la asignación de valores a las variables $\{X_1 = v_1,..., X_n = v_n\}$ tal que "v_i" pertenece al conjunto de dominios "D_i" y se verifican las restricciones.

En esta fórmula se permite una representación sencilla del problema y el uso de heurísticas de propósito general, independientemente del problema al que se le quiera aplicar.

A continuación se darán algunos ejemplos de algunos problemas de satisfacción de restricciones.

3.1.1 Ejemplo Coloreo de Mapas

Se requiere colorear un mapa con regiones adyacentes (como muestra la Figura 3.1) que además de compartir más de un punto, comparta todo un segmento de borde (frontera entre estas regiones) en común.

En algunas ocasiones solamente se requiere de tres colores para mapas más sencillos, pero en algunos casos se necesita un cuarto color adicional.

Para éste ejemplo se utilizará en particular el coloreo del mapa de Australia, usando los colores rojo, verde o azul, de tal forma que dos regiones vecinas no tengan el mismo color.

Figura 3.1 : Problema de Coloreo de Australia

Modelo:

- Las variables son las regiones de Australia AO(Australia del Oeste), TN(Territorio del norte), Q(Queensland), NGS(Nueva Gales del Sur), V(Victoria), AS(Australia del Sur), T(Tasmania).
- El dominio corresponde a los colores que van a tomar las distintas regiones de Australia, en este caso el conjunto {rojo, verde, azul}.
- Las restricciones requieren que regiones vecinas tengan distintos colores.

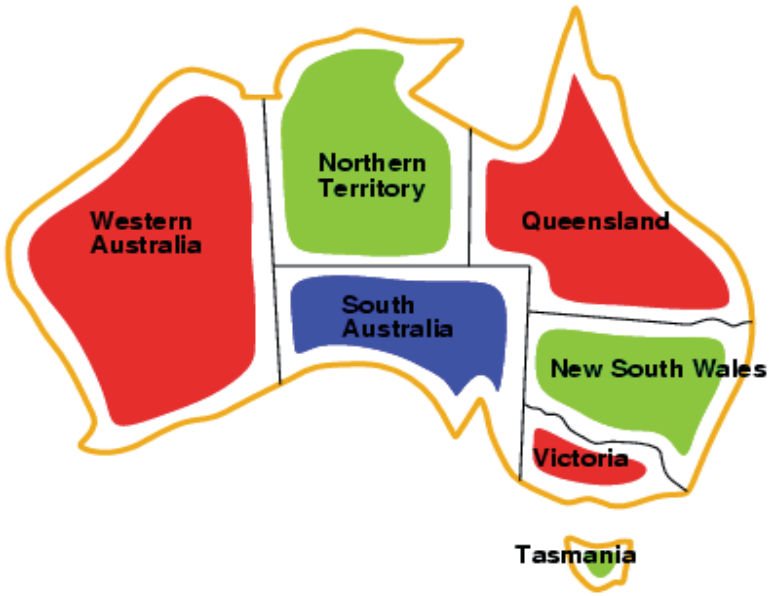

Figura 3.2 : Solución de Problema de Coloreo de Australia

3.1.2 Ejemplo N-Reinas

Este problema plantea que se deben posicionar N-Reinas en un tablero de ajedrez de NxN, de tal forma que las reinas no se puedan atacar entre ellas, para esto se debe considerar que una reina puede moverse de manera horizontal, vertical, diagonal y puede moverse en más de un espacio. Para este caso se utilizará N = 8.

Modelo:

- Variables: Para éste caso son 8 reinas: R_1, R_2, R_3, R_4, R_5, R_6, R_7, R_8.
- Dominio: El dominio corresponde al número de filas que tendrá el tablero, en este caso es [1,..,8].
- Restricciones: Las restricciones son no colocar dos reinas en una misma columna y en una diagonal. $i \in [1,7]$ y $j \in [i+1,8]$.

En la Figura 3.3 se muestra un tablero real de ajedrez con N = 8. Las reinas están posicionadas de tal manera que no puedan ser atacadas entre ellas y así dando una solución que satisfaga las restricciones anteriormente especificadas.

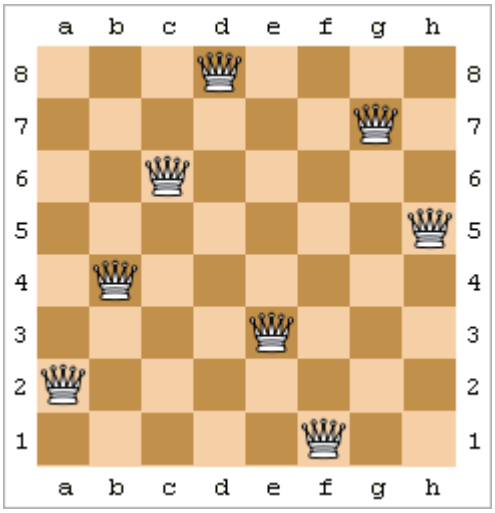

Figura 3.3 : Solución a problema de 8-Reinas

3.2 Algoritmos de Búsqueda

Los algoritmos de búsqueda realizan una búsqueda exhaustiva a través del espacio de todas las posibles asignaciones en búsqueda de resultados que cumplan de manera simultánea con todas las restricciones del problema, estos algoritmos realizan una búsqueda completa teniendo la seguridad de encontrar una solución o en caso contrario demostrando la no existencia de las mismas. La desventaja de estos algoritmos es que son muy costosos.

Estos algoritmos pueden ser llamados completos los cuales explotan todo el espacio de estados en búsqueda de la solución y se garantiza una solución, o también pueden ser incompletos si solamente exploran una parte del espacio de estados y su solución no se garantiza.

A continuación se describirán los métodos de búsqueda Generate and Test y Backtracking.

3.2.1 Generate and Test

Es un algoritmo de búsqueda sencillo, pero es ineficiente al encontrar todas las soluciones de un CSP. Un algoritmo Generate and Test (GT) genera de forma sistemática, todas las posibles asignaciones completas. Cuando termina de generar una asignación completa, comprueba si esta asignación es una solución (es decir, comprueba si satisface todas las restricciones). En terminología de árboles, GT es un algoritmo que recorre un árbol de búsqueda en profundidad prioritaria (recorrido primero en profundidad). La ineficiencia de GT se debe a que se genera muchas asignaciones completas que violan la misma restricción.

A continuación en la figura 3.4 se da un ejemplo del algoritmo Generate and Test aplicado en el problema de 4-reinas, en el cual se puede apreciar lo ineficiente que es al hacer todas las asignaciones.

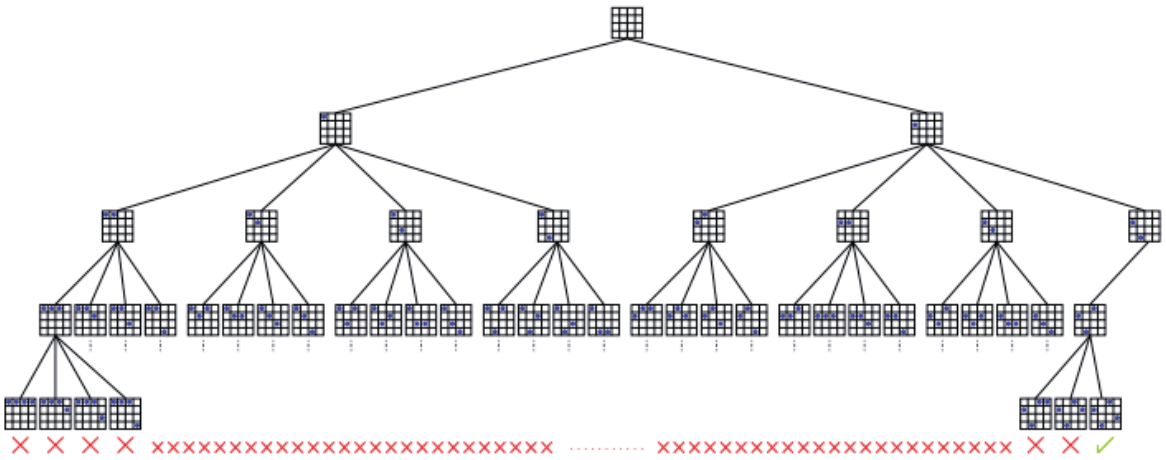

Figura 3.4 : Ejemplo de Algoritmo Generate and Test aplicado a 4-Reinas

3.2.2 Backtracking

Backtracking o también conocido como Backtracking Cronológico (BT) es una mejora del algoritmo Generate and Test (GT) y los aspectos en que éste es una mejora son: que al momento de asignar un nuevo valor a la variable actual (x_i), esta comprueba si hay consistencia con los valores asignados anteriormente. Si no es consistente, descarta esta asignación parcial y se asigna un valor nuevo a (x_i), si ya no hay más valores que asignar a D_i, BT retrocede para probar otros valores para la variable X_{i-1}, si no hay más valores para D_{i-1}, el algoritmo retrocede al nivel *i-2*, y así sucesivamente sigue iterando hasta encontrar un valor para una variable que satisfaga las restricciones, tomando en cuenta los valores de las variables pasadas.

La Figura 3.5 representa el proceso de búsqueda realizada por el procedimiento de backtracking (BT) en el problema 4-reinas. Aquí, podemos ver que BT es capaz de detectar los fallos tan pronto como dos variables se crean instancias (en el nivel medio del árbol), mucho antes que en el enfoque GT. A pesar de esto, el enfoque de BT no es capaz de detectar fallos antes de asignar los valores a todas las variables involucradas en un conflicto de restricción. Este problema puede ser manejado por la aplicación de consistencias locales.

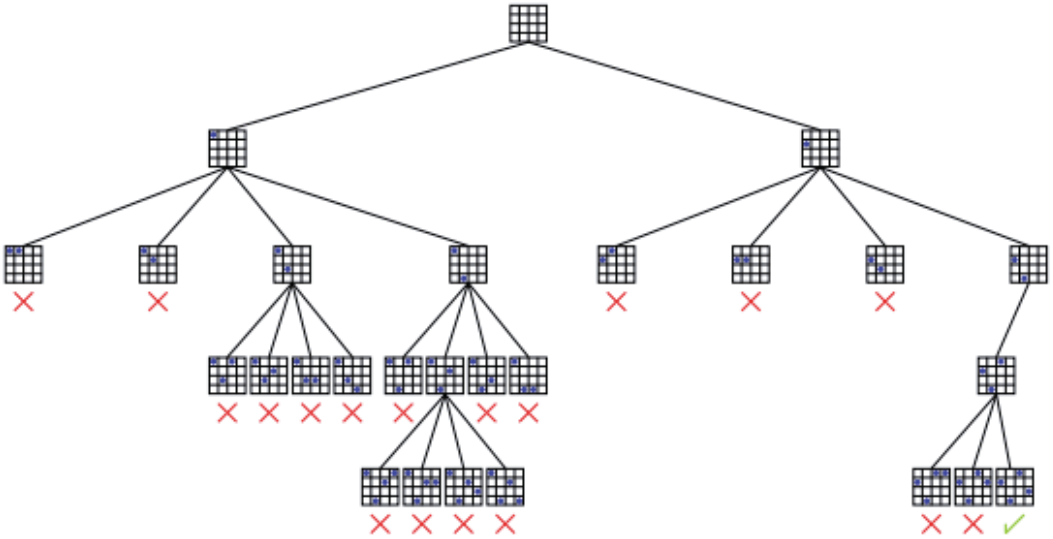

Figura 3.5 : Ejemplo de Algoritmo Backtracking para 4-Reinas

3.3 Técnicas de Consistencia

Las técnicas de consistencia o también conocidas como técnicas de inferencia, son utilizadas para mejorar la eficiencia de los algoritmos de búsqueda, éstas pueden ser utilizadas antes o mientras se hace el proceso de búsqueda, así reduciendo la cantidad de nodos a instanciar en el árbol de búsqueda, para esto se eliminan los dominios que debido a una restricción no corresponden a una solución.

3.3.1 Nodo-Consistencia

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

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

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

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

$$\forall X_i \in X, \forall R_i \in R: Xi \text{ satisface } R_i.$$

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

Al aplicar nodo consistencia se eliminarán del dominio los valores que no satisfacen la restricción unaria, por lo tanto el dominio quedaría dado por el conjunto {1, 2, 3, 4}, llamado dominio nodo-consistente.

3.3.2 Arco-Consistencia

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

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

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

Ejemplo: Dada una restricción $R_{ij:}\, x_i < x_j$ de la figura 3.6 se puede ver que el arco R_{ij} es consistente, ya que para cada valor de $a \in [2,5]$ hay al menos un valor en $b \in [7,9]$ de manera que satisface la restricción $R_{ij.}$

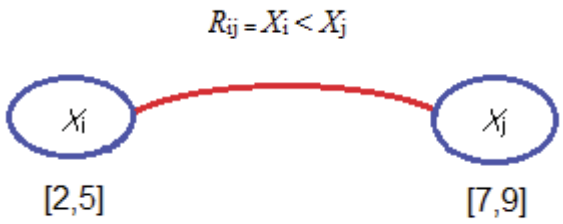

Figura 3.6 : Ejemplo de arco consistencia

3.4 Estrategias de Enumeración

La forma y el orden de la selección de variables cambian notablemente la eficiencia de la ejecución en la búsqueda de los algoritmos de búsqueda. Una buena mejora el tiempo en el que se soluciona un CSP se obtiene seleccionando un orden correcto de variables y valores. También puede ser importante un orden adecuado en las restricciones del problema, aunque este punto no tiene mucho estudio. A continuación se explicarán las heurísticas más importantes en la ordenación de variables y valores.

3.4.1 Heurísticas

Los algoritmos de búsqueda necesitan contar con un orden específico, el orden en que los valores del dominio son instanciados para cada variable y también se puede especificar el orden de las restricciones presentes en el problema. Una correcta selección del orden para la variable y valor, puede mejorar la eficiencia de la solución generada. Para determinar el orden es que existen las heurísticas de selección de variable y las heurísticas de selección de valor, las distintas combinaciones de estas heurísticas forman las estrategias de enumeración.

Cada una de las distintas estrategias de enumeración tiene un comportamiento distinto por lo que pueden existir algunas con un comportamiento más eficiente que otras.

3.4.1.1 Heurísticas de Selección de Variables

El orden de las variables son asignadas mientras la búsqueda pueda tener un efecto significativo en cuanto al espacio de búsqueda. Generalmente estas heurísticas intentan seleccionar con anterioridad las variables que más restringen a las demás, con el objetivo de reducir el número de vueltas atrás. Las heurísticas de selección de variables pueden ser estáticas o dinámicas.

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

3.4.1.1.1 Heurísticas de selección estática de variables

1) **Minimum Width (MW)**: Al comienzo impone un orden total sobre todas las variables, de forma que el orden tiene la mínima anchura (ancho de la red). La anchura de la variable *x* es el número de variables que están antes que *x*, de acuerdo a un orden dado y que son adyacentes a *x*. Así mismo la anchura de un orden es la máxima anchura de todas las variables bajo ese orden. En resumen las variables que se encuentran al principio de la ordenación son las más restringidas y las menos restringidas son las variables que se encuentran al final. Asignando antes las variables más restringidas, se pueden identificar antes las situaciones sin salida reduciendo el backtracking.
2) **Maximun Degree (MD)**: Las variables son ordenadas de forma decreciente según su grado en el grafo de restricciones original. El grado de un nodo hace referencia al número de nodos adyacentes a él, es decir el número de variables con la que está conectada. Debido al orden decreciente las variables de mayor grado (las más conectadas) son seleccionadas primero.
3) **Minimum Domain Variable (MDV)**: Selecciona las variables de acuerdo a la menor cardinalidad de su dominio. Por lo tanto aquellas variables cuyo dominio es pequeño son elegidas antes.

3.4.1.1.2 Heurísticas de selección dinámica de variables

1) **First-fail**: Primer fallo, sugiere que para tener éxito deberíamos intentar primero donde sea más probable que falle, entonces en cada paso se selecciona la variable más restringida. La heurística first-fail también conocida como mínimum remaining values (MRV), trata de hacer lo mismo seleccionando la variable con el dominio más pequeño, esto se sostiene en que si una variable posee un dominio pequeño es más difícil encontrar un valor consistente.
2) **Round Robin**: Se seleccionan las variables en algún orden racional e equitativo de esta manera se logran instanciar todas las variables presentes.

3.4.1.2 Heurísticas de Selección de Valor

En las heurísticas de selección elección de valor, se puede elegir, un valor que sea más probable de llevar a una solución reduciendo el riesgo de tener que hacer backtracking y probar un valor alternativo. En la práctica, lo mejor que se puede hacer es elegir un valor que es menos probable que nos conduzca a una falla.

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

2) **Max Domain Size:** Selecciona el valor que deja los máximos tamaños de dominio para las variables futuras.

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

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

3.5 Solvers

Los solvers que se utilizan para la programación con restricciones se basan en la programación declarativa, de este modo es necesario declarar las variables, los dominios pertinentes y las restricciones. Sólo con esos tres elementos el solver será capaz de encontrar una solución.

Distintos solvers ofrecerán distintos resultados, por ejemplo un solver puede obtener más rápido los resultados, o bien mostrar más robustez en comparación con otro.

3.5.1 ECL^{i}PS^e

El solver utilizado en la implementación de este trabajo es ECLiPSe por lo cual a continuación se describen algunas de sus principales características.

Es un sistema software libre, basado en el paradigma de la programación lógica con restricciones para el despliegue y desarrollo de aplicaciones de programación con restricciones. Es ideal para la enseñanza de la mayoría de los aspectos de un problema combinatorio, por ejemplo modelado de problemas, programación con restricciones, programación matemática y técnicas de búsqueda. Su amplio alcance lo hace ser una buena herramienta para la investigación de métodos de resolución de problemas híbridos.

Los principales libros de CLP (Constraint Logic Programming) hacen referencia al uso de ECLiPSe por lo cual se ha convertido en una herramienta importante en la programación con restricciones.

Está constituido por un núcleo en tiempo de ejecución, una colección de bibliotecas, modelado y control de lenguaje, un entorno de desarrollo, interfaces para integrarse en un entorno host e interfaces para solvers anexos.

ECL^iPS^e está diseñado para:

- Tareas generales de programación, especialmente para la creación rápida de prototipos.
- La resolución de problemas utilizando las librerías de resolución disponibles y el paradigma de la programación lógica con restricciones.
- El desarrollo de nuevos constraint solvers, basados en otros existentes.

ECL^iPS^e es en gran medida compatible con el lenguaje de programación lógico e interpretado Prolog (basado en tres tipos de cláusulas: hechos, reglas y consultas) y algunas extensiones que permitirán manejar bases de datos. El programa proporciona varias bibliotecas de constraint solvers las cuales pueden ser utilizadas en programas de aplicación:

- Además de enteros, la librería de dominios finito permite elementos atómicos, por ejemplo string y floats y también de elementos compuestos básicos por ejemplo f(a,b).
- Conjunto finito de restricciones.
- Restricciones lineales.
- Reglas de manejo de restricciones CHR (Constraint Handling Rules) que contiene bibliotecas de más de 20 constraint solvers adicionales.

4 Autonomous Search

Autonomous Search (AS) es un caso único de sistemas adaptativos, su objetivo es mejorar el rendimiento de la solución de un CSP, adaptándose a éste. Tiene la habilidad de transformar sus componentes internos cuando es expuesto a cambios externos y a oportunidades.

Su objetivo es mejorar el rendimiento de la solución, este se puede medir a través del valor de indicadores (backtracks, canditdad de nodos, etc) o del tiempo que demora en obtener la solución, para lograrlo se adapta la estrategia de búsqueda del problema. Los componentes internos, que se pueden modificar, corresponden a varios algoritmos involucrados en el

proceso de búsqueda: heurísticas, inferencias, etc. Los componentes externos corresponden a la evolución de la información recogida durante el proceso de búsqueda. Esta información también puede ser tomada directamente del problema (por ejemplo: tamaño del espacio de búsqueda exacto o estimado, número de sub-problemas) o indirectamente a través de la eficiencia vista de los componentes individuales, por ejemplo la capacidad de poda de las técnicas de inferencia.

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 elegidos, además posee una arquitectura donde sus componentes interactúan entre sí entregándose información, lo que sirve para el análisis de la siguiente estrategia a utilizar.

4.1 Arquitectura

Decidir las heurísticas de variable y valor a utilizar es un trabajo complejo, aun cuando estas sean elegidas, no se asegura de manera concreta el hecho de encontrar una solución para un problema de satisfacción de restricciones, o que si se encuentra la solución ésta sea la óptima. Dado esto, es necesario poner en marcha un proceso de resolución, donde será indispensable evaluar las estrategias para encontrar la solución.

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

Solver: Componente que corre un algoritmo CSP genérico, alternando la propagación de restricciones y las fases de enumeración. Posee un conjunto de estrategias de enumeración básicas, cada una de ellas caracterizada por una prioridad que evoluciona durante el cálculo, el componente actualización evalúa las estrategias y actualiza sus prioridades. Para cada enumeración (asignación un valor a una variable), la estrategia de enumeración dinámica selecciona la estrategia básica para ser usada en base a las prioridades fijadas.

Observación: Este componente tiene como objetivo registrar alguna información acerca del actual árbol de búsqueda, por ejemplo observar el proceso de resolución en el componente solver. Estas observaciones, llamadas snapshots, no son 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 estrategias y provee de indicadores para el componente actualización. Los indicadores pueden ser extraídos, calculados o deducidos desde uno o varios snapshots de la base de datos de snapshots.

Actualización: Componente que toma las decisiones usando una Choice Function, la cual determina el desempeño (o rendimiento) de una estrategia dada en una determinada cantidad de tiempo. Esta puede ser calculada basada en los indicadores entregados por el componente análisis.

4.2 Choice Function

Una Choice Function tiene como objetivo hacer un ranking con las múltiples estrategias de enumeración, dependiendo del rendimiento, según indicadores, que cada una de estas tiene. En otras palabras una Choice Function hace una comparación del rendimiento actual con su rendimiento histórico de las estrategias de enumeración.

Una Choice Function hace ranking, califica y luego elige entre las estrategias de enumeración. Para cualquier estrategia de enumeración S_j, la Choice Function f en el paso n por cada estrategia S_j está definida en la ecuación 1, donde l es el número de indicadores considerados y α es un parámetro para controlar la relevancia del indicador dentro de la Choice Function.

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

Ecuación Choice Function

5 Skyline

Existen un gran cantidad de problemas que pueden ayudar a tomar una decisión final con el método Skyline, estos problemas son conocidos por sus variables que no son directamente mejores que otras, es decir, existe más de una variable tentativa para determinar cuál podría ser la opción elegida. Dado esto, el ejemplo para explicar Skyline más comúnmente usado es el de los Hoteles, éste consta en suponer que se necesita alojamiento en una ciudad donde se está estudiando o tomando vacaciones por dar un ejemplo, existen muchas opciones de hoteles en la ciudad, pero el sujeto en cuestión busca una opción factible ya sea por precio bajo o distancia del centro educacional, estas dos variables se conoce que son inversamente proporcionales, mientras más cerca este el hotel de la universidad, más alto será el precio del mismo y viceversa, mientras más lejos esté, menor será el precio.

La elección de hoteles depende del criterio del sujeto involucrado, si está dispuesto a pagar más o sacrificar cercanía por ahorrar un poco de dinero. Pues aquí entra Skyline, es un cálculo que se realiza viendo qué puntos (hoteles) dominan a otros puntos, esto se le conoce como dominancia, al realizar dicha tarea, se tiene una serie de opciones tentativas a elegir donde se conoce que este conjunto de hoteles son mejores que los demás de la ciudad, a este conjunto de hoteles se les denomina el Skyline, el cual proporciona una mejor vista de opciones a elegir por el sujeto ayudando a su toma de decisiones [2].

De forma más técnica y formal, Skyline se define como los puntos que son dominados por otros puntos.

Definición: Espacio numérico *D* definido mediante un conjunto de dimensiones {d_1, d_2, d_3,...,d_d} y un conjunto de datos $S \in D$.

Un dato $p \in S$ puede ser representado como un punto p = {p[1], p[2], … , p[d]} donde p[i] es un valor de la dimensión d_i.

Dominancia: Un punto p = {p[1], p[2],…, p[d]} domina a otro punto q ={q[1], q[2],…, q[d]} si y solo si $p[i] \leq q[i]$ donde $1 \leq i \leq d$ y existe al menos una dimensión j tal que $p[j] < q[j]$.

Skyline (S) es el conjunto de puntos no dominado por otros puntos.

Ejemplo 1: En este ejemplo las dimensiones presentadas se derivan de la cantidad de indicadores que posee el problema, donde cada estrategia utilizada posee un comportamiento distinto para cada indicador, así se puede calcular Skyline de estas estrategias dependiendo del valor de sus indicadores.

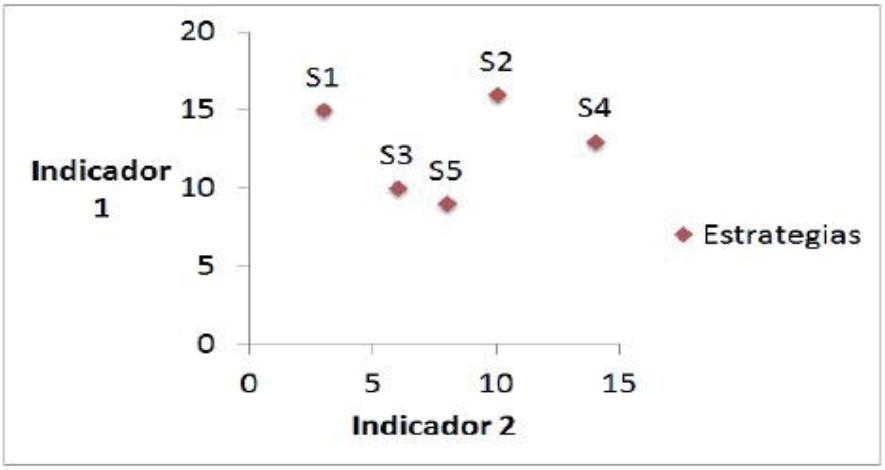

Figura 5.1 : Gráfico de Skyline de las Estrategias

En la figura 5.1 se puede ver los dos indicadores como las dimensiones del problema, en este caso indicador 1 y 2, mostrando los puntos en el plano que representan a las estrategias, de estos puntos es posible determinar la dominancia.

El principal objetivo de utilizar Skyline es obtener el grupo de puntos que sirvan para elegir la mejor estrategia y de estos puntos elegir el que tendrá mayor prioridad según criterios elegidos para la solución del problema en sí.

6 Algoritmos Skyline

Existen una cantidad limitada o pequeña de algoritmos que sirven para la resolución del Skyline, los que veremos a continuación serán el Block-Nested-Loops (BNL) y el Divide and Conquer, estos 2 algoritmos son completamente diferentes en la manera o metodología que usan para calcular el Skyline, BNL se basa en una ventana donde van ingresando tuplas dominantes, por otro lado el algoritmo Divide and Conquer, como dice su nombre, divide el espacio del problema donde están los datos con el objetivo de acotar el espacio de trabajo donde se detecta el Skyline [2]. A continuación se describe con mayor claridad y detalle el algoritmo BNL el cual se utiliza en el prototipo.

6.1 Algoritmo Block-Nested-Loops (BNL)

Block-Nested-Loops o BNL es bastante sencillo, se basa simplemente en la comparación de tuplas mediante algoritmos de ciclos anidados. Dicho algoritmo de ciclos anidados aplica a cada tupla la definición de dominancia descrita en el capítulo 5. El algoritmo también puede aplicarse en Skyline con más de 2 dimensiones pero se estaría presente a una falta de eficiencia de su parte. BNL es considerado rápido ya que produce tuplas Skyline en cada una de sus iteraciones, en lugar de considerar solo una tupla a la vez, este algoritmo lee de forma repetitiva un conjunto de tuplas [2]. El objetivo de esto es mantener la *ventana* de tuplas en memoria principal con la que trabaja BNL con tuplas incomparables. Cada vez que una tupla *x* se lee desde la entrada, se compara con las demás tuplas existentes en la *ventana* en busca de dominancia, según la comparación puede suceder 3 tipos de casos.

- *p* es dominado por una tupla ya existente en la *ventana*, en este punto para la comparación debido a que ya fue dominada por otra, *p* es eliminada y descartada de iteraciones futuras.

- *p* domina una o más tuplas pertenecientes a la *ventana*, dichas tuplas dominadas son eliminadas de la *ventana* y descartadas de iteraciones futuras. Luego la tupla *p* es insertada a la ventana.

- *p* es ingresado inmediatamente a la *ventana* debido a que no hay ninguna tupla con la cual establecer dominancia.

7 Top K en Skyline

El algoritmo Top-K implementado se basa en comparaciones de dominancia de las tuplas pertenecientes al Skyline, de esta manera selecciona la tupla con mayor dominancia entre ellas ayudando a seleccionar la mejor estrategia de ese momento. Dado esto, selecciona la mejor de las mejores teniendo así un mayor grado de confianza sobre la opción escogida aumentando el rendimiento en Autonomous Search.

8 Prototipo

8.1 Objetivo

Lo que inició la idea de llevar a cabo este prototipo Top-K fue la necesidad de buscar una manera de mejorar los tiempos de resolución de problemas CSP como por ejemplo N-Queen y Magic Square entre otros, con respecto a las estrategias de enumeración por si solas, Skyline, Random, etc. Los métodos anteriormente nombrados dan buenos tiempos de resolución de los CSP en cuestión, pero se llevó a cabo el nuevo método Top-K con el propósito de mejorar los tiempos de los mismos.

8.2 Funcionamiento

Se basa simplemente en un algoritmo Skyline, en este caso se utiliza Block-Nested-Loops para calcular el Skyline de las estrategias de enumeración, este algoritmo realiza una ventana de tamaño (tamVentana) $n/2$ si n es par o $(n/2)+1$ si n es impar, en donde n es la cantidad de estrategias a utilizar, en esta ventana se insertan las estrategias que dominan al resto o las que son incomparables, durante el avance de este algoritmo Skyline, se va contabilizando la cantidad de veces que un estrategia domina a otra, dando como resultado las veces exactas en que una estrategia dominó a otra mientras se aplica el algoritmo BNL. Una vez calculado el Skyline se selecciona las Top-K estrategias dentro de la ventana tomando como parámetro la cantidad de veces que una estrategia domino a otra, mientras más veces dominó a otra es mejor, en este caso se implementa el K=1 dado que solo interesa la mejor estrategia para que continúe la ejecución del Autonomous Search.

8.3 Algoritmo

Se calcula Skyline mediante el algoritmo BNL, mientras se ejecuta este algoritmo se trabajará con un arreglo de dominancia el cual llevara la cuenta de cuantas veces dominó cada estrategia, el momento en el que se realiza la cuenta es cuando la nueva tupla (estrategia) domina a la tupla en ventana, se aumenta el valor en 1 en la posición correspondiente a la tupla en el arreglo de dominancia, de la misma manera, cuando la tupla que está en la ventana domina a la nueva tupla entrante se aumenta el contador, cuando la estrategia entrante es incomparable con las que se encuentran en la ventana, no se suma el contador ya que no domina a ninguna estrategia. De esta manera se arma un arreglo de dominancia con la cantidad exacta de cuantas veces una estrategia dominó a otra (ver ilustración 8.1).

8.3.1 Código Algoritmo BNL

```
1: cargar_tuplas( );
2:
3: mientras tupla hacer
```

```
4:              si ventana_esta_vacia( ) entonces
5:                      insertar_en_ventana(tupla);
6:              sino
7:              si dominancia(tupla) entonces
8:                      borrar( );
9:                      insertar_en_ventana( );
10:                     contador_dominancia++; // gano la nueva tupla entrante
11:                     sino
12:                     eliminar_tupla( );
13:                     contador_dominancia++; // gano la tupla en ventana
14:                     fin_si
15:
16:                     si tupla_es_incomparable( ) entonces
17:                     insertar_en_ventana(tupla);
18:                     fin_si
19:             fin_si
20      fin_mientras
```

Ilustración 8.1: Algoritmo BNL

8.3.2 Código Algoritmo Top-K

Se crea un arreglo de dominancia con 2 filas y 8 columnas, la primera fila representa el número de estrategia y la segunda representa la cantidad de veces que dominó esa estrategia, se envía el arreglo de dominancia y la ventana vacía para calcular el Skyline descrito anteriormente, posteriormente se ordena el arreglo de dominancia de mayor a menor un algoritmo de ordenamiento, en este caso se utilizó el algoritmo Bubble Sort mejorado, una vez ordenado se compara la estrategia en el arreglo de dominancia con mayor valor (índice i=0) con las estrategias de la ventana, un vez encontrada la con mayor valor de dominancia, se selecciona como estrategia a utilizar y retorna. En el caso que la ventana posea solamente estrategias incomparables, estas no poseen valor de dominancia, por lo tanto, para resolver esta situación se elegirá la estrategia en ventana que tenga la menor cantidad de backtracks, si existe un empate en la cantidad de backtracks se seleccionara la estrategia en ventana que tenga la menor cantidad de nodos visitados y si aun así existe un empate en este indicador se elegirá una estrategia al azar.

```
1: double dominancia[2][8];
2: double ventana[tamVentana][cantIndicadores];
3: int flag = 0;
4:
5: calcularSkyline(dominancia, ventana);
6: ordenarDominancia(dominancia);
7:
8: desde i = 0 hasta 7
9:      desde j = 1 hasta tamVentana
10:             si (ventana[0][j] == dominancia[0][i]) Entonces
```

```
11:                         estrategia_a_utilizar = dominancia[0][i];
12:                         flag = 1;
13:                         i=8;
14:                         j=tamVentana;
15:                 fin_si
16:         fin_desde
17: fin_desde
18:
19: si(flag == 0) Entonces
20:         estrategia_a_utilizar = menor_backtrack(dominancia);
21:
22:         si(estrategia_a_utilizar  == 0) Entonces
23:                 estrategia_a_utilizar = menor_nodos_visitados(dominancia);
24:         fin_si
25:
26:         si(estrategia_a_utilizar  == 0) Entonces
27:                 estrategia_a_utilizar  = random();
28:         fin_si
29: fin_si
30:
31: retornar estrategia_a_utilizar;
```

Ilustración 8.2 : Algoritmo Top-K

9 Pruebas

Los experimentos realizados poseen las siguientes características:

- Sin conexión a internet.
- Sin otros programas abiertos.
- Se realizan 10 pruebas por problema y se escoge la mejor.

Características del PC:

- RAM: 4Gb.
- Disco Duro: 232 Gb.
- Procesador: Intel (R) Core (TM) i3-2120 CPU @ 3.30GHz
- Sistema Operativo: Windows 7 32 bits.

Las pruebas fueron realizadas bajo las características nombradas anteriormente, con la versión ECLiPSe 6.0, se seleccionaron todas las estrategias en la mayoría de los problemas, excepto donde se indicará posteriormente, no se realizó smoothing.

Todos y cada uno de los problemas(N-Queen, Magic Square, Knight Tour y Sudoku) se ejecutó 10 veces con cada una de las choice functions, con el objetivo de encontrar la mejor entre éstas. Primero se realizaron las pruebas solamente con Skyline más Trade Off, el cual le

da prioridad a ciertos indicadores sobre otros según la choice function utilizada, para realizar comparaciones con el nuevo método Skyline más Top-K que se desarrolló. El método Skyline más Trade Offs se utiliza para ayudar a elegir la mejor opción entre las estrategas de enumeración. Dado el tipo de solución es necesario que para cada choice function se ejecuten distintos Trade Offs los que se componen de los distintos indicadores presentes en la choice function, cada uno de estos se visualizan en la tabla 1. Para la choice function 1, los Trade Offs que se utilizaron para la elección de la mejor estrategia fueron aquellos donde se marca la preferencia de menor cantidad de backtrack en vez del número de step, esto debido a que se busca minimizar la cantidad de backtracks totales durante la ejecución del problema. La choice function 2 en sus trade offs marca una preferencia sobre la menor cantidad de shallow backtracks, que la variación de la máxima profundidad (*In1*), se marca esta preferencia debido a que los shallow bactrack no representan variación de profundidad porque sólo se instancian diferentes valores para la variable. En la choice function 4 ambos trade offs marcan preferencia sobre la mayor cantidad de variables fijadas. A diferencia de la última choice function donde se da prioridad a la menor cantidad de variables no fijadas, es decir aquellas estrategias que han fijado mayor cantidad de variables.

Choice Function		**Trade Offs**
CF_1	B-Step-PTAVFES	t_1 := B → Step
		t_2 := PTAVFES → B
CF_2	SB-In1-In2	t_1 := SB → In1
		t_2 := In1 → In2
CF_3	B-Step-In1-In2	t_1 := B → Step
		t_2 := In1 → In2
CF_4	VF-Dmax-DB	t_1 := VF → Dmax
		t_2 := VF → DB
CF_5	N-VU-VFPS-THRASH	t_1 := VU → VFPS
		t_2 := VU → THRASH

Tabla 9.1 : Choice Functions y Trade Offs utilizados.

A continuación en la tabla 9.2 se detallarán los tiempos de ejecución que resultaron en las pruebas con los problemas N-Queens, Magic Square, Knight Tour y Sudoku para todas las choice functions anteriormente descritas, resaltando cual fue la choice function que mejores resultados entregó (mejor tiempo de ejecución). La sigla *t.o*. significa time out, lo cual significa que la prueba demoró más de 1 día y no logró obtener una solución. Primero se analizará Skyline más Top-K y luego se realizará una comparación respecto a Skyline más Trade Off y Top-K por si solo, como primera observación se puede notar que la choice function 2 (SB-In1-In2) es la más efectiva y eficiente al momento de realizar las pruebas ya que fue la única en llegar a una solución para el problema de N-Queen para n igual 50 y 75 llegando a tiempos de 25,069 y 8,596 segundos respectivamente, para lograr estos resultados se excluyeron las estrategias 1 y 2 (ver Anexo A). Para los demás problemas y distintas n se utilizaron todas las estrategias. En el caso de *n* igual a 8 y 10 no se notaron cambios significativos en cuanto al tiempo de ejecución, también se observó que los resultados de *n*=20 fueron menores que a *n*=15 en las choice function 2, 3, 4 y 5 (ver tabla 1).

Para el problema de Magic Square se puede apreciar que hay diferencias significativas respecto a las pruebas con *n*=4 y *n*=5 en cuanto al tiempo de ejecución, luego para todas las choice function con *n*=4 no se notaron grandes diferencias (no más de 20 milisegundos entre cada una) por el contrario en n=5 la choice function 2 fue la que obtuvo el mejor resultado con un tiempo de 0,796 segundos. En el problema de Knight Tour tanto como para *n*=5 y *n*=6 no se llegó a un resultado en las choice función 1, 2, 3 y 4 (ver tabla 9.1), en la 5 se logró llegar un resultado con un tiempo de 32,557 y 44,858 segundos respectivamente. Cabe señalar que para obtener estos resultados se excluyeron las estrategias 1 y 2 (ver Anexo A). Por último no se apreciaron cambios significativos en el problema del sudoku obteniendo mejor runtime en las choice function 2, 3 y 4 con un tiempo de 0,093 segundos En la tabla 9.2, los campos en "*negrita*" significa que esa choice function fue la mejor con respecto al *n*, cabe destacar en el problema Sudoku, la que gano fue la CF5 con un runtime de 0,094 segundos el cual es 0,001 segundos más lento que las CF 2, 3 y 4, pero con valores de sus indicadores más bajos (backtracks, nodos visitados y steps).

	N-Queens							Magic Squares		Knight Tour		Sudoku
	n=8	n=10	n=12	n=15	n=20	n=50	n=75	n=4	n=5	n=5	n=6	Problem1
CF_1	0,100	0,100	0,156	0,745	0,828	t.o.	t.o.	0,109	9,017	t.o.	t.o.	0,094
CF_2	0,094	0,109	**0,125**	**0,440**	**0,406**	**25,069**	**8,596**	0,110	**0,796**	t.o.	t.o.	0,093
CF_3	**0,093**	0,094	0,125	0,740	0,718	t.o.	t.o.	**0,109**	6,146	t.o.	t.o.	0,093
CF_4	0,093	0,093	0,124	0,677	0,474	t.o.	t.o.	0,125	3,308	t.o.	t.o.	0,093
CF_5	0,109	**0,093**	0,141	0,815	0,530	t.o.	t.o.	0,141	2,184	32,557	44,858	**0,094**

Tabla 9.2 : Runtimes en segundos

La tabla que se verá a continuación (tabla 9.3) muestra la cantidad total de backtracks, nodos visitados y step para la mejor choice function de cada problema.

Para N-Queens la menor cantidad de backtracks se presentó en *n*=10 con 5 backtracks en total, por otro lado la prueba con mayor cantidad de backtracks se vio en *n*=50 con 5025 en total. Referente a la cantidad de nodos visitados, el menor fue 36 y la mayor fue 25368 para *n*=8 y *n*=50 respectivamente. En Step hubo un empate en *n*=8 y *n*=10 ambos con 15 step.

El problema Sudoku fue el con mejores resultados en cuanto a Backtracks (0 Backtracks), nodos visitados y Step, con respecto a Magic Square con *n*=4 y *n*=5; y Knight Tour con *n*=5 y *n*=6, este último problema con *n*=5 posee la mayor cantidad de Backtracks, Nodos visitados y Step con 8.499, 151.723 y 23.439 respectivamente.

	N-Queens							Magic Squares		Knight Tour		Sudoku
	n=8	n=10	n=12	n=15	n=20	n=50	n=75	n=4	n=5	n=5	n=6	Problem1
Backtracks	8	5	14	147	89	5025	1255	7	171	8499	7099	0
Nodos Visitados	36	39	87	743	470	25368	6601	67	1325	151723	138130	10
Steps	15	15	31	291	194	11446	2965	19	286	23439	23363	6
Choice Function	CF3	CF5	CF2	CF2	CF2	CF2	CF2	CF3	CF2	CF5	CF5	CF5

Tabla 9.3 : Cantidad total de backtracks, nodos visitados y Steps.

10 Análisis de Resultados.

Ya realizadas las pruebas y observando los resultados con las mejoras de desempate por backtracks y nodos visitados, en cuanto a backtracks, nodos visitados y steps en cada una de las pruebas, se puede llegar a un análisis de cómo se comportó Skyline más Top-K en la resolución de los problemas a tratar, tomando en cuenta las choice function utilizadas, indicadores, etc.

En los problemas donde el *n* total es un número pequeño véase N-Queen con *n*=8 y *n*=10, la solución con Top-K obtuvo muy buenos resultados en comparación al resto. Tiempo de ejecución bajos, disminuyendo en algunos casos los backtracks y nodos visitados, dado que en la ejecución el algoritmo Skyline se ejecuta muchas veces obteniendo así la mejor estrategia en ese mismo instante. En cuanto a los problemas con *n* de mayor tamaño, la alternativa Top-K posee una mayor cantidad de backtracks que ciertas estrategias por si solas, comúnmente MRV+ID y MRV+IDM, pero bastante menor que las demás estrategias, pese a esto, posee un runtime bastante cercano o incluso menor en ciertas pruebas con respecto a estas estrategias por si solas.

Analizando las choice function, se nota un mejor rendimiento según el tipo de problemas donde se utilizan, un ejemplo es la choice function 2 que fue la única en dar resultados con el problema N-Queens con *n*=50 y *n*=75, como también la choice function 5 que fue la única que llegó a una solución en el problema Knight Tour con *n*=5 y *n*=6, esto también con cierta configuración de estrategias ya dicho anteriormente. Para problemas como Magic Square con *n*=4, N-Queen con *n*= 8 y *n*=10 (con *n* pequeños); y Sudoku, no hubo una variación significativa de runtimes entre las 5 choice functions.

La cantidad de nodos visitados aumenta considerablemente en los problemas con *n* grandes, un ejemplo de esto es Knight Tour con *n*=5, el cual alcanzó los 151.723 nodos visitados, se puede decir que tanta cantidad de nodos visitados se debe a la gran cantidad de backtracks que posee este problema (8.499), al tener que instanciar una variable y que no pertenezca a la solución, se realiza el backtrack y así van sumando los nodos visitados, se puede decir que no fue una solución eficiente en el caso de Knight Tour ya que lo que se busca es disminuir lo más posible estos valores.

Los problemas N-Queens con *n*=50 y *n*=75; y Knight Tour con *n*=5 y *n*=6 presentaron time out en las choice functions 1, 3 y 4, esto ocurrió debido a que Skyline no elegía las mejores estrategias por lo cual entraba en un proceso de fijar variables con valores que no llegaban en ningún caso a la solución, es por esto que se desmarcó de la interfaz aquella (s) estrategia (s) que Skyline elegía y no eran las adecuadas para continuar la resolución del problema. Una vez solucionado esto, la solución se ejecutó correctamente solo en la choice function 2 en el caso de N-Queens y en la choice function 5 en el caso de Knight Tour, dando como resultado un tiempo considerablemente bueno para la complejidad de los problemas.

Las estrategias seleccionadas por el algoritmo Skyline más Top-K en el caso ganador de N-Queens con *n*=20 se muestran en la ilustración 10.1, este método para este caso en especial prefirió la estrategia 1 y 3 (F+ID y MRV+ID) por encima de las demás estrategias con un 32,77% y un 45,34% respectivamente, mientras que las demás estrategias van entre el 1% y el 7% como máximo. Las estrategias en el Skyline se prueban primero una por una para poder ver su comportamiento según los indicadores asociados a cada estrategia, al tener este conocimiento se comparan y según el criterio explicado anteriormente de Top-K (el con mayor cantidad de dominadas). Dependiendo del problema se nota cierta predilección por algunas estrategias que destacan entre el resto, esto significa que simplemente las más usadas son las con mejores valores en ese instante.

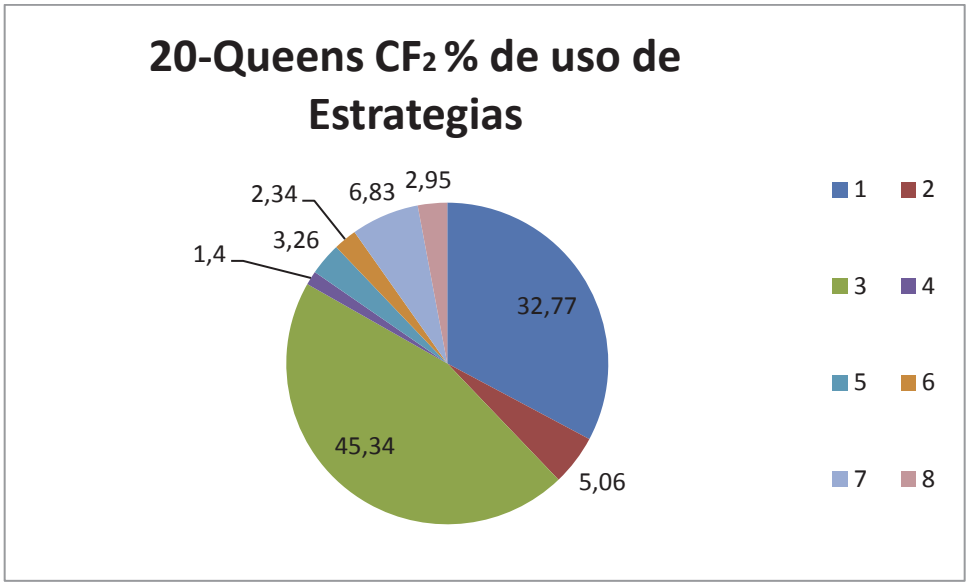

Ilustración 10.1 : Porcentaje de uso de estrategias

Los Backtracks, Nodos visitados y Steps en general fueron números bajos como también el tiempo de resolución de los problemas, en la mayoría de los casos fueron menores que en los casos de comparación, por lo que se puede aseverar que el método Skyline más Top-K entrega resultados satisfactorios con lo esperado.

11 Comparación de Resultados

Realizando pruebas con diferentes estrategias se obtienen variados resultados que sirven para realizar comparaciones entre los métodos para resolver un CSP como lo es Skyline más Top-K.

En la tabla 11.1 se muestra la cantidad de backtracks obtenidos en las pruebas realizadas con las diferentes estrategias que están en la herramienta, también con el método Top-K sin Skyline, Skyline con TradeOff y la nueva opción Skyline con Top-K.

En este problema la solución Skyline con Top-K no es ni la mejor ni la peor elección, se encuentra dentro del promedio, pero muestra una ligera ventaja sobre Trade Off con *n* pequeños en cuanto a backtracks, sigue obteniendo gran cantidad de este indicador cuando los *n* son mayores (*n*=50) pero sigue siendo bastante menor que la con mayor cantidad de backtracks (>39.232). Dado esto, si lo que se busca es minimizar la cantidad de backtracks, Skyline más Top-K es una buena opción para *n* pequeños.

Se destaca que en Sudoku hubo 0 backtracks y bajó significativamente la cantidad de éstos en Knight Tour con *n*=6, en los demás casos no es ni la menor ni la mayor, pero sigue estando en números más cercanos al menor valor lo que es bueno.

	N-Queen							Magic Squares		Knight Tour		Sudoku
Strategy	n=8	n=10	n=12	n=15	n=20	n=50	n=75	n=4	n=5	n=5	n=6	Problem1
F+ID	10	6	15	73	10026	>27406	>26979	12	910	767	>19818	18
AMRV+ID	11	12	11	808	2539	>39232	>36672	1291	>46675	>42889	>43098	10439
MRV+ID	10	4	16	1	11	177	818	3	185	767	>19818	4
O+ID	10	6	15	73	10026	>26405	>26323	10	5231	>18838	>19716	18
F+IDM	10	6	15	73	10026	>27406	>26979	51	>46299	767	>19818	2
AMRV+IDM	11	12	11	808	2539	>39232	>36672	42	>44157	>42889	>43098	6541
MRV+IDM	10	4	16	1	11	177	818	97	>29416	767	>19818	9
O+IDM	10	6	15	73	10026	>26405	>26323	29	>21847	>18838	>19716	2
Top-K	**5**	**6**	**0**	**13**	**122**	**5025**	**1255**	**1**	**207**	**12337**	**23756**	**2744**
Skyline Trade Off	**9**	**6**	**14**	**147**	**89**	**5025**	**1255**	**10**	**171**	**8499**	**12639**	**0**
Skyline Top-K	**8**	**5**	**14**	**147**	**89**	**5025**	**1255**	**7**	**171**	**8499**	**7099**	**0**

Tabla 11.1 : Backtracks de los problemas resueltos.

En la tabla 11.2 se muestran los nodos visitados en las pruebas realizadas, en el caso de N-Queens en los *n*=8 y *n*=10 aunque no obtuvo la menor cantidad de nodos visitados sino al contrario, es mayor que el resto, pero mostró una mejoría con respecto a Trade Off. En Sudoku se ve una cantidad mucho menor de nodos visitados (10 nodos) que las demás opciones, destacando la mejoría en cuanto a la técnica Top K sin Skyline en este problema.

	N-Queen							Magic Squares		Knight Tour		Sudoku
Strategy	**n=8**	**n=10**	**n=12**	**n=15**	**n=20**	**n=50**	**n=75**	**n=4**	**n=5**	**n=5**	**n=6**	**Problem1**
F+ID	24	19	43	166	23893	>65535	>65535	37	1901	3113	>65535	158
AMRV+ID	21	25	30	1395	4331	>65535	>65535	1826	>65535	>65535	>65535	30139
MRV+ID	25	16	45	17	51	591	2345	22	546	3113	>65535	76
O+ID	25	19	46	169	24308	>65535	>65535	31	13364	>65535	>65535	196
F+IDM	24	19	43	166	23893	>65535	>65535	110	>65535	3113	>65535	58
AMRV+IDM	21	25	30	1395	4331	>65535	>65535	69	>65535	>65535	>65535	19550
MRV+IDM	25	16	45	17	51	591	2345	230	>65535	3113	>65535	153
O+IDM	25	19	46	169	24308	>65535	>65535	61	>65535	>65535	>65535	62
Top-K	**26**	**39**	**7**	**83**	**660**	**25368**	**6601**	**22**	**1464**	**185067**	**428103**	**17275**
Skyline Trade Off	**46**	**41**	**87**	**743**	**470**	**25368**	**6601**	**85**	**1325**	**151723**	**217412**	**10**
Skyline Top-K	**36**	**39**	**87**	**743**	**470**	**25368**	**6601**	**67**	**1325**	**151723**	**138130**	**10**

Tabla 11.2 : Nodos Visitados de los problemas resueltos.

En la tabla 11.3 se muestran los runtimes en segundos, en N-Queens con *n*=8 y 10 Skyline con Top-K obtuvo los menores tiempos en general, solo superado por Top-K sin Skyline, en *n*=15 y *n*=20 no fue ni el peor ni el mejor, pero mostro una mejoría con respecto a Trade Off.

En Knight Tour con *n*=6 todas las estrategias mostraron t.o.(time out) menos en Skyline con Trade Off y con Top-K, siendo este último el que obtiene mejores resultados que Trade Off y Top-K sin Skyline. En el problema Sudoku obtuvieron los menores tiempos con Skyline más Top-K siendo la mejor opción para este problema.

	N-Queen							Magic Squares		Knight Tour		Sudoku
Strategy	**n=8**	**n=10**	**n=12**	**n=15**	**n=20**	**n=50**	**n=75**	**n=4**	**n=5**	**n=5**	**n=6**	**Problem1**
F+ID	0,125	0,125	0,156	0,296	35,354	t.o.	t.o.	0,140	2,919	4,306	t.o.	0,156
AMRV+ID	0,125	0,124	0,156	2,106	7,785	t.o.	t.o.	0,109	t.o.	t.o.	t.o.	6,271
MRV+ID	0,124	0,125	0,156	0,124	0,156	1,138	6,645	0,125	0,795	4,290	t.o.	0,141
O+ID	0,125	0,125	0,156	0,312	35,179	t.o.	t.o.	0,156	17,534	t.o.	t.o.	0,140
F+IDM	0,125	0,124	0,141	0,296	34,246	t.o.	t.o.	0,296	t.o.	4,305	t.o.	0,156
AMRV+IDM	0,125	0,125	0,140	2,137	7,785	t.o.	t.o.	0,187	t.o.	t.o.	t.o.	3,338
MRV+IDM	0,141	0,125	0,156	0,125	0,171	1,154	6,505	0,405	t.o.	4,352	t.o.	0,171
O+IDM	0,124	0,125	0,141	0,312	34,570	t.o.	t.o.	0,218	t.o.	t.o.	t.o.	0,187
Top-k	**0,004**	**0,007**	**0,003**	**0,018**	**0,265**	**23,041**	**9,286**	**0,006**	**0,405**	**21,575**	**86,997**	**6,349**
Skyline Trade Off	**0,095**	**0,100**	**0,120**	**0,527**	**0,485**	**25,023**	**9,812**	**0,119**	**0,815**	**49,260**	**87,597**	**0,100**
Skyline Top-K	**0,093**	**0,093**	**0,125**	**0,440**	**0,406**	**25,069**	**8,596**	**0,109**	**0,796**	**32,557**	**44,858**	**0,093**

Tabla 11.3 : Runtimes de los problemas resueltos.

12 Conclusiones

Para resolver todos los problemas planteados en esta investigación y muchos más que no están presentes en este documento se requiere una gran base de conocimiento teórico y práctico en el área estudiada como es Autonomous Search y sus derivados, ya sea Programación con Restricciones, Problemas de Satisfacción de Restricciones, Skyline, Algoritmos de Búsqueda, etc.

Dado el creciente interés de este tema en los últimos años de parte de la comunidad informática, resulta interesante aprender e investigar sobre sus áreas como el Skyline, el cual ayuda significativamente a la toma de decisiones de variados problemas, siendo uno de los más conocidos el típico ejemplo de los hoteles ya descrito anteriormente.

Se revisaron dos algoritmos para calcular el Skyline, Block-Nested-Loops (BNL) y Divide and Conquer de los cuales se utilizó Block-Nested-Loops para realizar dicha tarea. Luego de calcular el Skyline, se calcula la dominancia que posee cada estrategia de enumeración para realizar comparaciones entre ellas utilizando un algoritmo de ordenamiento (Bubble Sort Mejorado) y determinar el Top One (la mejor) de las estrategias de enumeración en la resolución de un CSP.

Se puede decir que la implementación del algoritmo Top-K a la programación con restricciones y Autonomous Search dado los buenos resultados obtenidos en esta investigación motivan a seguir indagando en el tema.

Los resultados obtenidos por el prototipo Skyline más Top-K en la gran mayoría de los problemas con las diferentes choice function comparado con los resultados del prototipo Trade Off lograron menores runtimes, backtracks, nodos visitados y steps. Cabe destacar que en algunos casos Trade Off igualó o supero los resultados de Skyline más Top-K dependiendo del problema y de las choice function escogidas. Con respecto a esto, en algunos casos en que los 2 métodos se igualan en backtracks, nodos visitados y steps, Skyline más Top-K muestra un menor runtime que Trade Off, lo que da a entender que la eficiencia del algoritmo Top-K es ligeramente mejor que la del algoritmo Trade Off. Un ejemplo de lo que anteriormente se ha dicho es el caso de N-Queens con un n=20 con la choice function 2, ambos métodos (Top-K y Trade Off) dieron como resultado los mismos números en todos sus indicadores, pero Skyline más Top-K dio un runtime de 0,406 segundos mientras que Trade Off alcanzo los 0,485 segundos.

El algoritmo actual ha ido evolucionando ya que poseía algunos problemas en la contabilización de las estrategias dominantes, como también al aplicar cierta configuración de estrategias que se requerían para que ciertas pruebas entregaran resultados sin time out (véase Capitulo 9), como lo son Knight Tour con n=5 y n=6; además de N-Queens con n=50 y n=75.

Se notó una mejoría en ciertos problemas como n-reinas y magic square al implementar el desempate por backtracks y nodos visitados, como también hubo casos en que los valores fueron levemente mayores a los con desempate al azar.

13 Referencias

[1] Xuemin Lin, Yidong Yuan, Qing Zhang, Ying Zhang. **Selecting Stars: The k Most Representative Skyline Operator Conference**, 2007.

[2] Stephan Borzsonyi, Donald Kossmann, Konrad Stocker. **The Skyline Operator**, 2001.

[3] http://progrxrestric.blogspot.com/. Visto última vez el 26 de diciembre 2013.

[4] http://www-sop.inria.fr/coprin/cgrandon/Publica/grandon2004.pdf. Visto última vez el 26 de diciembre 2013.

[5] Federico Barber, Miguel A. Salido. **Introducción a la Programación de Restricciones**, 2002.

[6] http://es.wikipedia.org/wiki/Programación_con_restricciones. Visto última vez el 26 de diciembre 2013.

[7] http://ccc.inaoep.mx/~jagonzalez/ADA/AlgVor.pdf. Visto última vez el 26 de diciembre 2013.

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

[9] http://es.wikipedia.org/wiki/NP-hard. Visto última vez el 26 de diciembre 2013.

[10] http://research.microsoft.com/apps/pubs/default.aspx?id=70590. Visto última vez el 26 de diciembre 2013.

[11] http://es.wikipedia.org/wiki/Algoritmo_ heurístico. Visto última vez el 26 de diciembre 2013.

ANEXO

A: Indicadores

Código	Nombre	Descripción	Cálculo
Step	Número de Mediciones	Cada vez que se usa la choice function.	Step++
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.	length(AllVars,N)
VU	Variables no Instanciadas	Número de variables no instanciadas.	var_count(Rest,VU)
SB	Shallow Backtrack	Cuando se trata de asignar un valor a la variable actual sin éxito entonces se recorre el próximo valor.	
SS	Espacio de Búsqueda	Actual espacio de búsqueda.	$\prod (Dom_i)_t$
SS_pr	Anterior Espacio de Búsqueda	Espacio de búsqueda anterior.	$\prod (Dom_i)_{t-1}$
B	Backtracks	Cuando la variable actual lleva a una ruta sin salida, entonces de retrocede a la variable anterior.	Cada vez que suceda el contador aumenta en uno.
N	Nodos	Cantidad de nodos visitados.	El contador se incrementa en uno cada vez que se visita un nodo.

Tabla A1: Indicadores Base

VF	Variables Instanciadas	Número de variables instanciadas por enumeración y propagación.	(TV - VU)
VFE	Variables Fijadas por Enumeración en cada Paso	Funciona cuando el número de pasos es mayor que 1.	1
TVFE	Número Total de VFE	Número total de variables instanciadas.	$\sum$ VFE
VFP	Variables Fijadas por Propagación en cada Paso.	Número de variables instanciadas por propagación.	length(Rest,N) - VU
TVFP	Número Total de VFP	Número total de variables instanciadas por propagación.	$\sum$ VFP
TSB	Total Shallow Backtracks	Número total de Shallow Backtracks.	$\sum$ SB
d_pr	Profundidad previa	Profundidad previa en el árbol de búsqueda.	$(TV - VU)_{t-1}$
D	Profundidad	Profundidad actual en el árbol de búsqueda.	$(TV - VU)_t$
dmax_pr	Profundidad máxima previa	La profundidad máxima previa en el árbol de búsqueda.	$MAX(dmax)_{n-1}$
In1	Profundidad máxima actual – Profundidad máxima previa	Representa la variación de la profundidad.	(dmax - dmax_pr)
In2	Profundidad actual – profundidad previa.	Si es positivo significa que el nodo actual se encuentra a más profundidad que el del paso previo.	$(d_t - d_{t-1})$
In3	Reducción del espacio de búsqueda.	Si es positivo, el actual espacio de búsqueda es más pequeño que el del Snapshot anterior.	$((SS_{pr} - SS)/ SS\_pr) \times 100$
PVFE	Porcentaje de VFE		(VFE/TV) x 100
PVFET	Porcentaje de las variables totales instanciadas por enumeración en cada paso		(TVFE/TV) x 100

PVFP	Porcentaje de variables instanciadas por propagación en cada paso		(VFP/TV) x 100
PVFPT	Porcentaje de las variables totales instanciadas por propagación en cada paso		(TVFP/TV) x 100
Thrash	Thrashing		$(d_{t-1} - VFPS_{t-1})$
B_real		Si la variable actual lleva a una ruta sin salida, entonces se va a la variable previa y cuenta ese retroceso.	Si D-d_pr==0 entonces Incrementar Si D < d_pr entonces d_pr – D + 1

Tabla A2: Indicadores Calculados

B: Estrategias de Enumeración

	Heurística De Selección Variable	**Heurística De Selección Valor**
Estrategia 1	First	Indomain
Estrategia 2	AMRV	Indomain
Estrategia 3	MRV	Indomain
Estrategia 4	Occurence	Indomain
Estrategia 5	First	Indomain Max
Estrategia 6	AMRV	Indomain Max
Estrategia 7	MRV	Indomain Max
Estrategia 8	Occurence	Indomain Max

Tabla B1: Estrategias de enumeración presentes en la herramienta.

Nombre	**Descripción**
First	Escoge la primera variable de la lista.
AMRV	Escoge la variable con el dominio más grande.
MRV	Escoge la variable con el dominio más pequeño.
Occurence	Escoge la variable con mayor cantidad de restricciones.

Tabla B2: Heurística de Selección de Variables.

Nombre	**Descripción**
Indomain	Escoge el elemento más pequeño del dominio.
Indomain Max	Escoge el elemento más grande del dominio.

Tabla B3: Heurísticas de Selección de Valores.

C: Pruebas Skyline más Trade Off

N-Queens							
	n=8	**n=10**	**n=12**	**n=15**	**n=20**	**n=50**	**n=75**
Backtracks	9	6	14	147	89	5025	1255
Nodos Visitados	46	41	87	743	470	25368	6601
Steps	18	16	31	291	194	11446	2965
Choice Function	CF_5	CF_1	CF_2	CF_2	CF_2	CF_2	CF_2

Tabla C1: Cantidad total de backtracks, nodos visitados y Steps. Problema N-Queens.

	Magic Squares		**Knight Tour**		**Sudoku**
	n=4	**n=5**	**n=5**	**n=6**	**Problem1**
Backtracks	10	171	8499	12639	0
Nodos Visitados	85	1325	151723	217412	10
Steps	22	286	23439	39457	6
Choice Function	CF_1	CF_2	CF_5	CF_5	CF_5

Tabla C2: Cantidad total de backtracks, nodos visitados y Step. Problema Magic Square, Knight Tour, Sudoku.

D: Promedios Runtime y Comparación

Información

Delta	Mejor SkylineTopK - (el mejor entre SkylineTradeOff o TopK)
x	No hay runtime por Time Out

Tabla D1: Nomenclatura de las tablas.

N-Reinas

N=8	CF1	CF2	CF3	CF4	CF5
Mejor SkylineTopK	0,100	0,094	0,093	0,093	0,109
Peor SkylineTopK	0,145	0,140	0,125	0,125	0,141
Promedio SkylineTopK	0,120	0,111	0,108	0,106	0,114
Skyline TradeOff	0,100	0,095	0,100	0,105	0,095
Top K	0,004	0,090	0,007	0,008	0,009
Delta	0,096	0,004	0,086	0,085	0,100

Tabla D2 RunTime promedio y comparación de N-Queen con n=8.

N=10	CF1	CF2	CF3	CF4	CF5
Mejor SkylineTopK	0,100	0,109	0,094	0,093	0,093
Peor SkylineTopK	0,120	0,110	0,125	0,109	0,125
Promedio SkylineTopK	**0,107**	**0,109**	**0,1108**	**0,101**	**0,103**
Skyline TradeOff	0,100	0,105	0,100	0,105	0,100
Top K	0,013	0,110	0,007	0,013	0,008
Delta	0,087	0,004	0,087	0,08	0,085

Tabla D3: RunTime promedio y comparación de N-Queen con n=10.

N=12	CF1	CF2	CF3	CF4	CF5
Mejor SkylineTopK	0,156	0,125	0,125	0,124	0,141
Peor SkylineTopK	0,218	0,171	0,187	0,203	0,218
Promedio SkylineTopK	**0,170**	**0,133**	**0,151**	**0,151**	**0,173**
Skyline TradeOff	0,120	0,120	0,125	0,135	0,130
Top K	0,003	0,100	0,020	0,018	0,011
Delta	0,153	0,025	0,105	0,106	0,13

Tabla D4: RunTime promedio y comparación de N-Queen con n=12.

N=15	CF1	CF2	CF3	CF4	CF5
Mejor SkylineTopK	0,745	0,440	0,740	0,677	0,815
Peor SkylineTopK	1,369	1,000	1,187	1,128	1,207
Promedio SkylineTopK	**0,953**	**0,668**	**0,959**	**0,885**	**0,974**
Skyline TradeOff	0,585	0,527	0,550	0,665	0,550
Top K	0,018	0,120	0,062	0,088	0,052
Delta	0,727	0,320	0,678	0,589	0,763

Tabla D5: RunTime promedio y comparación de N-Queen con n=15.

N=20	CF1	CF2	CF3	CF4	CF5
Mejor SkylineTopK	0,828	0,406	0,718	0,874	0,530
Peor SkylineTopK	1,337	0,842	1,092	1,233	1,061
Promedio SkylineTopK	**1,080**	**0,650**	**0,885**	**1,080**	**0,841**
Skyline TradeOff	0,950	0,485	0,922	0,610	0,550
Top K	0,265	0,300	0,378	0,316	0,286
Delta	0,563	0,106	0,340	0,558	0,244

Tabla D6: RunTime promedio y comparación de N-Queen con n=20.

N=50	CF1	CF2	CF3	CF4	CF5
Mejor SkylineTopK	x	25,069	x	x	x
Peor SkylineTopK	x	28,064	x	x	x
Promedio SkylineTopK	x	**26,509**	x	x	x
Skyline TradeOff	x	25,023	x	x	x
Top K	x	23,041	x	x	x
Delta	x	2,028	x	x	x

Tabla D7: RunTime promedio y comparación de N-Queen con n=50.

N=75	CF1	CF2	CF3	CF4	CF5
Mejor SkylineTopK	x	8,596	x	x	x
Peor SkylineTopK	x	10,140	x	x	x
Promedio SkylineTopK	x	**9,479**	x	x	x
Skyline TradeOff	x	9,812	x	x	x
Top K	x	9,286	x	x	x
Delta	x	-0,690	x	x	x

Tabla D8: RunTime promedio y comparación de N-Queen con n=75.

Magic Square

N=4	CF1	CF2	CF3	CF4	CF5
Mejor SkylineTopK	0,109	0,110	0,109	0,125	0,141
Peor SkylineTopK	0,156	0,156	0,125	0,140	0,188
Promedio SkylineTopK	**0,1266**	**0,1341**	**0,1186**	**0,1373**	**0,1718**
Skyline TradeOff	0,119	0,122	0,133	0,124	0,144
Top K	0,006	0,093	0,021	0,083	0,023
Delta	0,103	0,017	0,088	0,042	0,118

Tabla D9: RunTime promedio y comparación de Magic Square con n=4.

N=5	CF1	CF2	CF3	CF4	CF5
Mejor SkylineTopK	9,017	0,796	6,146	3,308	2,387
Peor SkylineTopK	11,061	1,155	7,488	4,087	3,151
Promedio SkylineTopK	**9,986**	**0,928**	**0,6080**	**0,3549**	**0,2693**
Skyline TradeOff	8,967	0,815	9,624	3,241	3,405
Top K	15,201	0,405	8,774	0,572	10,023
Delta	0,050	0,391	-2,628	2,736	-1,018

Tabla D10: RunTime promedio y comparación de Magic Square con n=5.

Sudoku

	CF1	CF2	CF3	CF4	CF5
Mejor SkylineTopK	0,094	0,093	0,093	0,093	0,094
Peor SkylineTopK	0,110	0,110	0,110	0,110	0,110
Promedio SkylineTopK	**0,102**	**0,101**	**0,103**	**0,100**	**107,6**
Skyline TradeOff	0,110	0,100	0,105	0,105	0,100
Top K	x	13,369	x	6,349	14,911
Delta	-0,016	-0,007	-0,012	-0,012	-0,006

Tabla D11: RunTime promedio y comparación de Sudoku.