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

ASIGNACIÓN DE ACCIONES COMERCIALES A GRUPOS DE CLIENTES UTILIZANDO CONSTRAINT PROGRAMMING

ERIC MICHAEL AGUILERA ARCE

TESIS DE GRADO
MAGÍSTER EN INGENIERÍA INFORMÁTICA

JULIO 2014

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

ASIGNACIÓN DE ACCIONES COMERCIALES A GRUPOS DE CLIENTES UTILIZANDO CONSTRAINT PROGRAMMING

ERIC MICHAEL AGUILERA ARCE

Profesor Guía: **Dr. Broderick Crawford Labrin**

Programa: **Magíster en Ingeniería Informática**

JULIO 2014

Resumen

Cada vez la información de Clientes es más valiosa para las empresas y para los Casinos de Juegos no es una excepción, dado que toda la información que se adquiere cuando un Cliente se enrola en el Club de Fidelización podría ser utilizada para realizar acciones particulares a éstos, con el fin de mejorar el servicio y darle un trato preferencial. En el presente trabajo se aborda cómo podrían seleccionarse distintas acciones aplicables a los clientes optimizando los costos incurridos en estas. Para lograr el objetivo se comparan dos técnicas de Optimización resolviendo el problema del "Set Partitioning", en base a una fuente de datos de pruebas existente, para luego aplicarlos al problema presentado.

Palabras Claves: Set Partitioning Problem, Constraint Programming, Metaheurísitcas, Algoritmos Genéticos, Cruce, Mutación, Grupos de Clientes, Acciones, Optimización de Costos.

Abstract

Each time, the information provided by clients is more valuable to all kind of business, and field of the industry of Gaming Casinos are not the exception. All the information obtained, from a customer when he is enrolled in a Customer Loyalty Club, could be used to take personalized private actions on them, in order to improve the services and to offer a preferential treatment. This work addresses how different actions applicable to customers could be selected to optimize the costs incurred in these, to achieve the objective, two optimization techniques are compared solving the problem of "Set Partitioning", based on a source existing test data, and then to apply them to the problem presented.

Keywords: Set Partitioning Problem, Constraint Programming, Metaheuristic, Genetic Algorithms, Crossing, Mutation, Customer Groups, Actions, Cost Optimization.

Índice

Tablas

Figuras

1 Introducción

En la actualidad el uso de la tecnología y nuevas técnicas de resolución para problemas con cierto grado de complejidad cada día se hace más frecuente. Tomando en cuenta además, que el poder de procesamiento se incrementa de manera sustancial con el correr del tiempo. La investigación y utilización de nuevas maneras de resolver problemáticas se ha ido transformando en un aliado cada vez más potente para las compañías que buscan como objetivo la minimización de costos y la búsqueda de nuevos indicadores que les permitan conocer más a sus clientes.

Ninguna empresa de consumo o servicios escapa al análisis de información relevante de sus clientes o potenciales compradores, es por lo mismo que deben estar en continuo proceso de búsqueda para lograr el procesamiento adecuado de la información, con esto las compañías podrían tomar directrices tanto para abarcar nuevos segmentos de clientes o simplemente desechar ciertas acciones y cambiar de estrategia de negocio.

El presente trabajo tiene por objetivo entregar un análisis de dos técnicas para resolver una problemática presente en una compañía de entretenimiento.

En primera instancia estas técnicas se evalúan por separado en base a la resolución de un conjunto de datos preparado específicamente para analizar la performance de estas con el fin de sintonizar los parámetros de cada técnica, para que presente el mejor desempeño en la resolución de la problemática, basado en el tiempo, memoria de computo utilizada y resultados entregados.

Una vez que cada técnica fue configurada para obtener su mejor performance, se comparan entre ellas en base a los mismos indicadores nombrados anteriormente. Con esto y con pruebas más contundentes se toma una decisión de seleccionar sólo una técnica que se aplicará al problema real.

Ya con la técnica seleccionada. Esta se ejecuta sobre datos reales entregados por la compañía y se genera una información de salida, la cual se entrega al área de CRM de la compañía para que haga uso del resultado del procesamiento y pueda ejecutar una campaña piloto para evaluar el resultado entregado por esta técnica.

2 Definición de Objetivo general y específicos

2.1 Objetivo General

Optimizar la asignación de acciones comerciales predefinidas a grupos de Clientes del Casino pertenecientes al club de fidelización en base al costo de estas.

2.2 Objetivos Específicos

- Modelamiento del Problema en base al Set Partitioning Problem.
- Evaluación de Constraint Programming y Algoritmos genéticos en base a la resolución del Set Partitioning Problem para resolver la problemática de asignación de Acciones a Grupos de Clientes basándose en el resultado de los experimentos.
- Entregar como resultado las asignaciones de Acciones Comerciales a Grupos de Clientes con los menores costos posibles.

3 Formulación del Problema

En la actualidad la Empresa Enjoy S.A busca como objetivo de negocio lograr que sus productos y servicios sean consumidos por la mayor cantidad de gente posible que visita las instalaciones. Es importante tomar en cuenta que los servicios prestados por esta organización (la gran mayoría) sólo pueden ser consumidos por personas mayores de 18 años de edad, en base a la legislación vigente.

Dentro de los productos la cadena ofrece servicios de hotelería, eventos, juegos de azar, bares y restaurant los cuales para consumirlos requiere necesariamente la presencia del cliente, ya que de otra manera sería imposible en la actualidad. Por ende, la probabilidad de que los servicios sean consumidos aumenta mientras más personas sean atraídas a las instalaciones donde se entregan estos servicios.

Es por lo mismo que el área encargada de trabajar con el club de fidelización ejecuta acciones que buscan como objetivo atraer a los clientes a las dependencias del casino de juegos. Una de estas actividades es generar acciones comerciales sobre los clientes, las se traducen en regalo de entradas, tragos, vales por juego, etc.

Este proyecto busca como fin entregar una herramienta que permita luego de analizar los clientes, y la generación de grupos de comportamiento, definir qué acciones deben ejecutarse sobre ciertos grupos de fidelización con el fin de minimizar los costos de estas asignaciones,

La idea es lograr que el cliente se sienta lo más cómodo y satisfecho posible con el servicio entregado, ya que al conocer de mejor manera al cliente se puede tratar de manera más personalizada y con esto dar una atención que lo lleve a volver a las instalaciones por una nueva experiencia de servicio.

Si bien, en la actualidad esta problemática puede ser abordada mediante un análisis con herramientas básicas, como por ejemplo a través del cruzamiento de información en planillas Excel, u otras más avanzadas como análisis de resultados obtenidos luego de consultas a base de datos y otros métodos que permitan realizar tratamiento de información, ese de nuestro interés abordarlo con herramientas y metodologías que nos permitan realizar optimizaciones a fin de buscar las mejores opciones de resolución.

Cuando se habla del tratamiento de grupos de clientes, se dejan fuera ciertos grupos que son minoritarios por definición, los que incluyen a clientes de alto valor, y por ende es inviable generar acciones masivas a estos tipos de clientes ya que estas son definidas de manera más particular.

En la actualidad, se definen grupos de clientes los cuales poseen ciertas características y por otro lado se generan acciones que aplican a ciertos grupos de clientes. Estas acciones están relacionadas a la entrega gratuita se servicios a los clientes, los cuales podrán hacer uso de estos bajo ciertos criterios de validez, con esto se generan campañas, las cuales representan un grupo de acciones que son aplicables a los clientes.

En la actualidad se presenta la problemática de la asignación de productos o servicios a clientes que podrían ser menores en relación a los costos, e incluso con la dualidad de asignaciones lo que se traduce en un aumento de costos (gastos) para la organización al momento de ejecutar la campaña comercial.

Como la cobertura de columnas a filas, para este problema en particular, basada en asignación de acciones a grupos de clientes, se asemeja a un problema ya resuelto por varias técnicas, incluyendo además que ningún grupo puede estar asociado a más de una acción a la vez, restricción que permite aún más minimizar los gastos realizados en la campaña, lo cual lleva a tomar la decisión de realizar un modelamiento en base al Set Partitioning Problem y abordar la optimización mediante la utilización de dos técnicas, Constraint Programming y Algoritmos Genéticos.

3.1 Set Partitioning Problem

El problema del Set Partitioning, también conocido por sus siglas SPP pertenece a la familia de los problemas polinomiales no determinísticos duros, el cual es un problema clásico en las Ciencias de la computación [1], y consiste en encontrar el número mínimo de conjuntos que contengan un cierto número de elementos de todos los conjuntos. En otras palabras, consiste en encontrar un conjunto de soluciones que permitan cubrir un conjunto de necesidades al menor costo posible. Un conjunto de necesidades corresponde a las filas, y el conjunto solución es la selección de columnas que cubren en forma óptima al conjunto de filas. Es importante destacar que cada fila puede estar cubierta si y sólo si por una única columna [2]. Este problema puede ser resuelto por una cantidad de algoritmos desarrollados a lo largo del tiempo. En este trabajo se utilizan dos técnicas en particular, programación con restricciones y algoritmos genéticos.

El modelo matemático del Set Partitiong es como sigue [3]:

Sea:

$A = (a_{ij})$ una matriz binaria (0, 1) de dimensiones m x n

$C = (C_j)$ un vector n dimensional de enteros.

$M = \{1, \dots, m\}$, representa las filas de la matriz

$N = \{1, \dots, n\}$, representa las columnas de la matriz

El valor $c_j (j \in N)$ representa el costo de la columna j, y podemos asumir, sin pérdida de generalidad, $c_j > 0$ para $j \in N$. Así decimos que la comuna $j \in N$ cubre la fila $i \in M$ si $a_{ij} = 1$

La función Objetivo es la siguiente:

$$Min \sum_{j=1}^{n} c_j x_j \qquad (1)$$

Sujeto a:

$$\sum_{j=1}^{n} a_{ij}\, x_j = 1 \quad \forall\, i \in M \qquad (2)$$

$x_j = \begin{cases} 1 \\ 0 \end{cases}$ $\forall\, j \in N$, x_j puede tomar los valores 0 ó 1, dependiendo si se activa o no la columna que cubre la fila.

3.2 Programación con restricciones

La programación por restricciones es una metodología de software utilizada para la descripción y posterior resolución efectiva de cierto tipo de problemas, típicamente combinatorios y de optimización. Estos problemas aparecen en muy diversas áreas, incluyendo inteligencia artificial, investigación operativa, bases de datos y sistemas de recuperación de la información con aplicaciones en scheduling, planificación, razonamiento temporal, diseño en la ingeniería, problemas de empaquetamiento, criptografía, diagnóstico, toma de decisiones. Estos problemas pueden modelarse como problemas de satisfacción de restricciones (Constraint Satisfaction Problems - CSP) y resolverse usando técnicas de satisfacción de restricciones. En general, se trata de grandes y complejos problemas, típicamente de complejidad NP. Las etapas básicas para la resolución de un problema CSP son su modelización y su posterior resolución mediante la aplicación de técnicas CSP específicas, que incluyen procesos de búsqueda apoyados con métodos heurísticos y procesos inferenciales [4].

Muchas decisiones que tomamos a la hora de resolver diversos problemas cotidianos están sujetas a restricciones. Decisiones tan cotidianas como fijar una cita, planificar un viaje, comprar un automóvil o preparar un plato de cocina puede depender de muchos aspectos interdependientes e incluso conflictivos, cada uno de los cuales está sujeto a un conjunto de restricciones que se deben satisfacer para que la decisión sea válida. Además, cuando se encuentra una solución que satisface plenamente a unos criterios, puede que no sea tan apropiada para otros, por lo que, para obtener una solución optimizada, no suele ser suficiente con obtener una única solución. Los primeros trabajos relacionados con la programación de restricciones datan de los años 60 y 70 en el campo de la Inteligencia Artificial. Durante los últimos años, la programación de restricciones ha generado una creciente expectación entre expertos de muchas áreas debido a su potencial para la resolución de grandes y complejos problemas reales. Sin embargo, al mismo tiempo, se considera como una de las tecnologías menos conocida y comprendida. La programación de restricciones se define como el estudio de sistemas computacionales basados en restricciones. La idea de la programación de restricciones es resolver problemas mediante la declaración de restricciones sobre el dominio del problema y consecuentemente encontrar soluciones a instancias de los problemas de dicho dominio que satisfagan todas las restricciones y, en su caso, optimicen unos criterios determinados.

3.2.1 Definiciones y conceptos básicos

La definición clásica de CSP radica en buscar la primera solución que satisfaga a todo el conjunto de restricciones. Esta definición clásica puede ampliarse para representar problemas de optimización, problemas cuyo enfoque es la búsqueda de la mejor solución (s) para un criterio dado. En el marco de satisfacción de restricciones, los problemas de optimización se denominan problemas de optimización restricción (COP), que se puede definir por una 4-tupla $\langle X, D, C, O \rangle$ donde O es una función objetivo que corresponde al criterio dado para ser minimizado o maximizado.

El CSP en general también puede especializarse en otras categorías, por ejemplo, en función de los valores de los dominios. Como un ejemplo, un dominio finito CSP significa un CSP que incluye únicamente valores enteros, un CSP numérico se refiere a CSP en reales. Otros ejemplos son los CSP booleanos y los CSPs simbólicos, esta última considera dominios no numéricos [5].

Los conceptos clave en esta metodología corresponden a los aspectos de:

- La modelización del problema, que permite representar un problema mediante un conjunto finito de variables, un dominio de valores finito para cada variable y un conjunto de restricciones que acotan las combinaciones válidas de valores que las variables pueden tomar. En la modelización de CSP, es fundamental la capacidad expresiva, a fin de poder captar todos los aspectos significativos del problema a modelar.

- Técnicas de consistencia o inferenciales que permiten deducir nueva información sobre el problema a partir de la explícitamente representada. Estas técnicas también permiten acotar y hacer más eficiente el proceso de búsqueda de soluciones.

- Técnicas de búsqueda de la solución, apoyadas generalmente por criterios heurísticos, bien dependientes o independientes del dominio. El objetivo es encontrar un valor para cada variable del problema de manera que se satisfagan todas las restricciones del problema. En general, la obtención de soluciones en un CSP es NP-completo, mientras que la obtención de soluciones optimizadas es NP-duro, no existiendo forma de verificar la optimalidad de la solución en tiempo polinomial. Por ello, se requiere una gran eficiencia en los procesos de búsqueda

Estos algoritmos se basan en la exploración sistemática del espacio de soluciones hasta encontrar una solución o probar que no existe tal. Las técnicas de consistencia o inferenciales permiten deducir información del problema, esto resulta sumamente útil ya en combinación con las técnicas de búsqueda, reducen el espacio de soluciones.

3.2.1.1 Modelización

Generalmente la declaración de un problema se suele expresar de muchas maneras diferentes, e incluso en lenguaje natural. Una parte muy importante para la resolución de problemas es el modelado del problema en términos de CSPs, es decir, variables, dominios y restricciones.

En la modelización CSP, resulta fundamental poder disponer de un modelo de restricciones adecuado para la especificación del problema. Con un problema mal modelado será imposible encontrar las soluciones que se requieren.

Las restricciones pueden agruparse según su aridad que es el número de variables que componen dicha restricción:

- Una restricción unaria es una restricción que consta de una sola variable.
- Una restricción binaria es una restricción que consta de dos variables.
- Una restricción ternaria consta de tres variables.
- Una restricción no binaria (n-aria) es una restricción que involucra a un número arbitrario de variables.

Ejemplo: La restricción $x \leq 3$ es una restricción unaria sobre la variable x. La restricción $x_1 - 3x_3 \neq 2$ es una restricción binaria. La restricción $2x_1 - 3x_2 + 5x_3 \geq 2$ es una restricción ternaria. Por último un ejemplo de restricciones n-aria sería $x_1 + 2x_2 - x_3 + 5x_4 \leq 9$.

Una tupla p de una restricción $C_{i..k}$, es un elemento del producto cartesiano D_i $x..x$ D_k. Una tupla p que satisface la restricción $C_{i..k}$, se le llama tupla permitida o válida. Una tupla p que no satisface la restricción $C_{i..k}$, se le llama tupla no permitida o no válida. Una tupla p de una restricción $C_{i..k}$, se dice que es soporte para un valor $a \in D_j$ si la variable $x_j \in X_{c_{1..k}}$ es una tupla permitida y contiene a a en la correspondiente posición de x_j en la restricción.

3.2.1.2 Técnicas de consistencia o inferenciales

Una de los principales problemas que presentan los algoritmos de búsqueda es la aparición de inconsistencias locales que van emergiendo continuamente. Las inconsistencias locales son valores individuales o combinación de valores de las variables que no pueden participar en la solución porque no satisfacen alguna propiedad de consistencia.

Las técnicas de consistencia local mejoran la eficiencia de los algoritmos de búsqueda al borrar valores inconsistentes de las variables o inducen restricciones implícitas que ayudan a podar el espacio de búsqueda. Estas técnicas de consistencia local se usan como etapas de preproceso donde se detectan y se eliminan las inconsistencias locales antes de empezar o durante la búsqueda con la finalidad de reducir el árbol de búsqueda.

Ejemplo. Si el valor a de la variable j es incompatible con todos los valores de una variable y pendiente de asignación, ligada a x mediante una restricción, entonces a es inconsistente y no formará parte de ninguna solución del problema. Por lo tanto, si se fuerza alguna propiedad de consistencia local, es posible borrar todos los valores que son inconsistentes con respecto a dicha propiedad.

Puede haber valores que son consistentes con respecto a un cierto nivel de consistencia local pero son inconsistentes con respecto a cualquier otro nivel mayor de consistencia local. Así, consistencia global asegura que todos los valores de los dominios de las variables que no pueden participar en una solución son eliminados.

Las restricciones implícitas, aquellas que se generan por la combinación de las restricciones explícitas (conocidas en el modelo), generan inconsistencias locales. Si un algoritmo de búsqueda no almacena las restricciones implícitas, repetidamente redescubrirá la inconsistencia local causada por ellas y malgastará esfuerzo de búsqueda tratando repetidamente de intentar instanciaciones que ya han sido probadas.

Ejemplo: Si se tiene un problema con tres variables x, y, z , con los dominios $\{0,1\},\{2,3\},\{1,2\}$ respectivamente. Hay dos restricciones en el problema: $y < xyz \neq y$. La búsqueda mediante Backtracking trata de instanciar las variables en el orden x, y, z , entonces probará todas las posibles 2^3 combinaciones de valores para las variables antes de descubrir que no existe solución alguna. Si observa la restricción entre yyz es posible ver que no hay ninguna combinación de valores para las dos variables que satisfagan la restricción. Si el algoritmo pudiera identificar esta inconsistencia local antes, se evitará un gran esfuerzo de búsqueda.

Niveles de Consistencia Local

Consistencia de Nodo (1-consistencia)

Es el nivel de consistencia local más simple de todas. En la consistencia de nodo o nodo-consistencia todos los valores en el dominio de una variable satisfacen todas las restricciones unarias sobre esa variable. Así, un problema es nodo-consistente si y sólo si todas sus variables son nodo-consistentes:

$$\forall x_i \in X, \forall C_i, \exists a \in D_i : a \text{ satisface } C_i$$

Ejemplo: Si se considera una variable x en un problema con dominio $\{2,15\}$ y la restricción unaria $x \leq 7$. La consistencia de nodo eliminará el intervalo $\{8,15\}$ del dominio de x. En la Figura 4 se muestra el resultado de aplicar nodo-consistencia a la variable x.

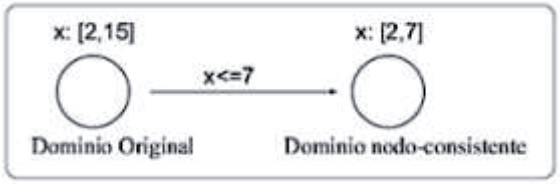


Figura N° 3.2.1: Consistencia de nodo (nodo-consistencia)

Consistencia de Arco (2-consistencia)

Una de las técnicas de consistencia más difundida y utilizada para reducir dominios en CSP con dominios discretos es la arco-consistencia [6].

Un problema binario es arco-consistente si para cualquier par de variables restringidas x_i y x_j, para cada valor a en D_i hay al menos un valor b en D_j tal que las asignaciones (x_i, a) y (x_i, b) satisfacen la restricción entre x_i y x_j. Cualquier valor en el dominio D_i de la variable x_i que no es arco-consistente puede ser eliminado de D_i ya que no puede formar parte de ninguna solución. El dominio de una variable es arco-consistente si todos sus valores son arco-consistentes.

Ejemplo: Dada la restricción $C_{ij} = x_i < x_j$ de la Figura 5, es posible observar que el arco C_{ij} es consistente, ya que para cada valor $a \in [3,6]$ hay al menos un valor $b \in [8,10]$ de manera que se satisface la restricción C_{ij}. Sin embargo si la restricción fuese $C_{ij} = x_i > x_j$ no será arco-consistente.

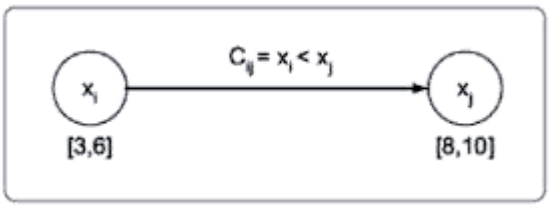


Figura N° 3.2.12: Consistencia de arco (arco-consistencia)

Así, un problema es arco-consistente si y sólo si todos sus arcos son arco-consistentes:

$\forall C_{ij} \in C, \forall a \in D_i, \exists b \in D_j$, tal que b es un soporte para a en C_{ij}.

Consistencia de Caminos (3-consistencia)

La consistencia de caminos es un nivel más alto de consistencia local que la arco-consistencia. La consistencia de caminos requiere para cada par de valores a y b de dos variables x_i y x_j, que la asignación de a a x_i y de b a x_j satisfaga la restricción entre x_i y x_j, y que además exista un valor para cada variable a lo largo del camino entre x_i y x_j de forma que todas las restricciones a lo largo del camino se satisfagan.

$$\forall (a,b) \in C_{ij}, \forall x_k \in X, \exists c \in D_k, \text{ tal que } c \text{ es un soporte para } a \text{ en } c_{ik} \text{ y para } b \text{ en } c_{jk}.$$

Ejemplo: Dado el problema representado mediante el grafo de restricciones de la Figura 6, se puede observar que el problema satisface la consistencia de caminos ya que cualquier camino entre un par de nodos lo es.

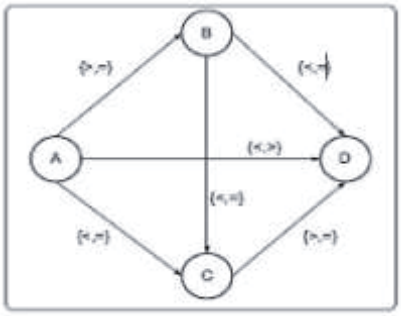


Figura N° 3.2.13: Consistencia de caminos

Cuando un problema satisface la consistencia de caminos y además es nodo-consistente y arco-consistente se dice que satisface fuertemente la consistencia de caminos (strongly path consistent).

Consistencia Global

La consistencia global es una noción más fuerte que la consistencia local y sus niveles. En CSP definido como globalmente consistente si solo contiene valores que combinados forman parte de al menos una solución.

Dado un CSP (X, D, C), se dice que es globalmente consistente si y sólo si $\forall x_i \in X, \forall a \in D_i, x_i = a$ forma parte de una solución del CSP.

Por tanto, un CSP globalmente consistente contiene solamente aquellas combinaciones de valores que forman parte de al menos una solución, según esto, se desprende que la red nunca admite una combinación de valores que no desemboque en una solución lo que implica que todas las soluciones están representadas en él. En una red de restricciones globalmente consistente la búsqueda puede llevarse a cabo sin la utilización de algoritmos de búsqueda.

3.2.1.3 Técnicas de Búsqueda

Los algoritmos de Backtracking son la base fundamental de los algoritmos de búsqueda para la resolución de CSPs. Estos algoritmos buscan a través del espacio de las posibles asignaciones de valores a las variables garantizando encontrar una solución, si es que existe, o demostrando que el problema no tiene solución, en caso contrario [7]. Es por ello por lo que se conocen como algoritmos completos. Los algoritmos incompletos, que no garantizan encontrar una solución, también son muy utilizados en problemas de satisfacción de restricciones y en problemas de optimización debido a su mayor eficiencia y el alto costo que requiere una búsqueda completa. Estos algoritmos incluyen algoritmos genéticos, tabú search, colonia de hormigas, entre otros.

Técnicas de búsqueda sistemática

Generar y Testear (GT)

Este algoritmo de búsqueda es la manera más sencilla, pero menos eficiente, de encontrar todas las soluciones de un CSP. Este método genera las posibles tuplas de instanciación de todas las variables de forma sistemática y después testea sucesivamente sobre cada instanciación si se satisfacen todas las restricciones del problema. La primera combinación que satisfaga todas las restricciones, será la solución al problema [8].

La poca eficiencia de GT se debe a que genera muchas asignaciones completas que violan la misma restricción. El número de combinaciones posibles es el producto cartesiano de todos los dominios de las variables. Es posible mejorar este algoritmo extendiendo una solución parcial incrementalmente con la asignación a una variable que sea consistente con la solución parcial obtenida hasta ese momento. Este tipo de resolución es conocida como Backtracking cronológico.

Backtracking Cronológico

Este algoritmo trabaja de la siguiente manera. El algoritmo selecciona la siguiente variable de acuerdo al orden de las variables y le asigna su próximo valor. Esta asignación de la variable se comprueba en todas las restricciones en las que forma parte la variable actual y las anteriores. Si todas las restricciones se han satisfecho, el Backtracking cronológico selecciona la siguiente variable y trata de encontrar un valor para ella de la misma manera.

Si alguna restricción no se satisface entonces la asignación actual se deshace y se prueba con el próximo valor de la variable actual. Si no se encuentra ningún valor consistente entonces existe una situación sin salida (dead-end) y el algoritmo retrocede a la variable anteriormente asignada y prueba asignándole un nuevo valor. Si se busca una única solución, el Backtracking cronológico finaliza cuando todas las variables se les han asignado un valor, en cuyo caso devuelve una solución, o cuando todas las combinaciones de variable-valor se han probado sin éxito, en cuyo caso no existe solución.

En la siguiente figura se describe el proceso de Backtracking aplicado a la resolución de CSP, donde V_n representa el vector de asignaciones a las variables $(x_1, x_2 ... x_n)$ del problema, ordenadas según un determinado criterio. La función $(K, V_{[n]})$ comprueba la validez de las k instanciaciones ya realizadas $(V_1, V_2 ... V_k)$, es decir, si se satisfacen las restricciones locales existentes entre las variables $(x_1, x_2 ... x_k)$.

Procedimiento backtracking (k; $V_{[n]}$); Llamada inicial : backtracking (1; $V_{[n]}$);

Inicio

$V_{[n]}$ = seleccionar (d_k) /* Selecciona un valor de d_k para asignar a x_k */

Si comprobar (k; $V_{[n]}$) Entonces

Si $k = n$ Entonces

Devolver $V_{[n]}$ /* Es una solución */

Sino

backtracking$(k + 1; V_{[n]})$

Fin Si

Sino

Si quedan_valores($d_{[k]}$) Entonces

backtracking$(k; V_{[n]})$

Sino

Si $k = 1$ Entonces

Devolver 0 /* Falló */

Sino

backtracking$(k - 1; V_{[n]})$

Fin Si

Fin Si

Fin Si

Fin

Fin backtracking

Figura N° 3.2.11: Algoritmo Backtracking Cronológico

Este algoritmo puede hacer bastante trabajo innecesario, si la variable que provoca la vuelta atrás del algoritmo no es precisamente la última asignada, haciendo que se tengan que probar todas las combinaciones posibles de las variables que hay entre el primer punto donde se encuentra la violación de la restricción y la variable que la provoca. El problema radica en que el Backtracking tiene una visión local del problema. Sólo comprueba restricciones que están formadas por la variable actual y las pasadas, e ignora la relación entre la variable actual y las futuras. Es posible optimizar Backtracking añadiendo al algoritmo los siguientes puntos:

Mejorar la función comprobar: Añadiendo a la función que compruebe la validez de la instanciación parcial efectuada respecto a las variables pendientes de asignar, permitiría detectar cuanto antes si la solución parcial efectuada de las k variables forma o no parte de una solución global, evitando profundizar en una secuencia de asignaciones que no conduzca a una solución. Esto se realiza mediante un proceso de propagación de las instanciaciones parciales ya realizadas al resto de la red.

Establecer un orden de instanciación de las variables: Mejorando el orden de evaluación de las variables (orden del vector $V[n]$) o estableciendo un criterio para la selección de los valores de los dominios correspondientes (seleccionar (d_k)) permitirían mejorar la búsqueda evitando los backtracks, por el hecho de seleccionar valores o variables que sean identificadas a prori como factibles de instanciar al no provocar inconsistencias.

Para ayudar a solucionar estos problemas, se han desarrollado algunos algoritmos de búsqueda más robustos. Estos algoritmos se pueden dividir en algoritmos Look-Backward y Look-Ahead y pertenecen a la categoría de las Técnicas de búsqueda híbridas.

Técnicas Hibridas

Algoritmos Look-Backward

Una de los problemas del Backtracking era el comportamiento al momento de enfrentar las situaciones sin salida. Los algoritmos Look-Backward tratan de explotar la información del problema para comportarse más eficientemente. Al igual que el Backtracking Cronológico, los algoritmos Look-Backw ard llevan a cabo la comprobación de la consistencia hacia atrás, es decir, entre la variable actual y las pasadas. Algunos algoritmos Look-Back.

Backjumping

Es un algoritmo para CSPs parecido al Backtracking Cronológico excepto que se comporta de una manera más inteligente cuando encuentra situaciones sin salida. En vez de retroceder a la variable anteriormente instanciada, Backjumping salta a la variable más profunda (más cerca de la variable actual) x_j que está en conflicto con la variable actual x_i donde $j < i$. Decimos que una variable instanciada x_j está en conflicto con una variable x_i si la instanciación de x_j evita uno de los valores en x_i (debido a la restricción entre x_j y x_i). Cambiar la instanciación de x_j puede hacer posible encontrar una instanciación consistente de la variable actual. Esto consigue evitar todas las pruebas de valores entre la variable actual y esa variable [9].

Conflict-directed Backjumping

Tiene un comportamiento de salto hacia atrás más sofisticado que Backjumping. Cada variable x_i tiene un conjunto conflicto formado por las variables pasadas que están en conflicto con x_i . En el momento en el que la comprobación de la consistencia entre la variable actual x_i y una variable pasada x_j falla, la variable x_j se añade al conjunto conflicto de x_i . En una situación sin salida, Conflict-directed Backjumping salta a la variable más profunda en su conjunto conflicto, por ejemplo x_k , donde $k < i$. Al mismo tiempo se incluye el conjunto conflicto de x_i al de x_k , por lo que no se pierde ninguna información sobre conflictos. Obviamente, Conflict-directed Backjumping necesita unas estructuras de datos más complejas para almacenar los conjuntos conflicto [10].

Ejemplo. Se pretende colorear el mapa de Australia, usando los colores rojo, verde o azul, de tal forma que dos regiones vecinas no tengan el mismo color.

Formulando el problema mediante CSP, quedaría con las siguientes variables: AO (Australia del Oeste), TN para Territorio del Norte, Q para Queensland, NGS para Nueva Gales del Sur, V para Victoria, AS para Australia del Sur y T para Tasmania, siendo el dominio para cada variables los colores rojo, verde y azul y como única restricción la diferencia de color entre regiones vecinas.

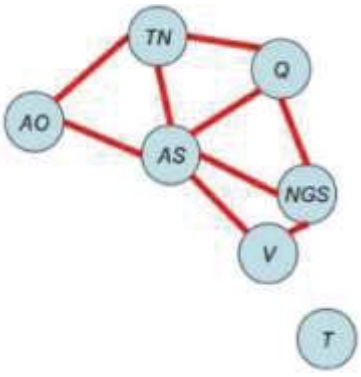
Figura A	Figura B
	

Figura N° 3.2.12: Algoritmo Backtracking Cronológico

Si se realizará la siguiente asignación de valores a las variables: {Q=rojo, NGS=verde, V=azul, T=rojo}. Cuando se intentara asignar la siguiente variable, AS, todo fallaría. Backtracking cronológico no solucionaría el problema ya que regresar y cambiar el valor a T no mejoraría la situación. Un enfoque más inteligente que el BT es regresar hasta una de las variables del conjunto que causaron la falla. Este conjunto es llamado el conjunto conflicto.

El conjunto conflicto para AS es {Q, NGS, V}. Conflict-direct backjumping regresaría a la variable más reciente en el conjunto conflicto. En este caso, saltaría sobre Tasmania y buscaría un nuevo valor para V.

Algoritmos Look-Ahead

Como ya se indicó en el punto anterior, los algorit mos Look-Backward tratan de reforzar el comportamiento de Backtracking mediante un comportamiento más inteligente cuando se encuentran en situaciones sin salida. Sin embargo, todos ellos todavía llevan a cabo la comprobación de la consistencia solamente hacia atrás, ignorando las futuras asignaciones de las variables [11].

Por cada nueva instanciación, los algoritmos Look-A head realizará una comprobación inferencial hacia adelante, para ello integran técnicas de consistencia o inferenciales durante el proceso de búsqueda, por lo que se denominan técnicas hibridas. Por cada etapa de búsqueda, los algoritmos Look-Ahead hacen una comprobación hacia adelante en cada etapa de la búsqueda llevando a cabo las comprobaciones para obtener las inconsistencias de las variables futuras involucradas además de las variables actual y pasadas. De esa manera, las situaciones sin salida se pueden identificar antes y además los valores inconsistentes se pueden descubrir y eliminar para las variables futuras. Esto también se conoce como la propagación de los efectos de cada nueva instanciación al resto de la red. Ello permite:

- Acotar las restricciones y dominios de las variables futuras a instanciar, limitando el espacio de búsqueda pendiente.

- Encontrar las inconsistencias antes de que aparezcan, en el caso de que las instanciaciones parciales efectuadas se descubran inconsistentes con el resto de variables pendientes. En definitiva, intentan descubrir si la actual asignación localmente consistente de las k variable s puede ser extendida a una solución global, provocando en caso contrario un punto de backtracking.

Forward Checking

Es uno de los algoritmos Look-Ahead más comunes [10]. Forward Checking, por cada etapa de la búsqueda, comprueba hacia adelante la asignación actual con todos los valores de las futuras variables que están restringidas con la variable actual. Si en esta búsqueda encuentra inconsistencias, los valores de las futuras variables que son inconsistentes con la asignación actual son temporalmente eliminados de sus dominios. Si el dominio de una variable futura se queda vacío, la instanciación de la variable actual se deshace y se prueba con un nuevo valor. Si ningún valor es consistente, entonces se lleva a cabo el Backtracking Cronológico.

Forward Checking garantiza que, en cada etapa, la solución parcial actual es consistente con cada valor de cada variable futura. Además cuando se asigna un valor a una variable, solamente se comprueba hacia adelante con las

futuras variables con las que están involucradas. Así mediante la comprobación hacia adelante, Forward Checking puede identificar antes las situaciones sin salida y podar el espacio de búsqueda.

El algoritmo sería idéntico al que hemos visto para el backtracking cronológico, salvo que incluiríamos la reducción de dominios mediante la prueba de arco consistencia tras hacer la asignación de valor a la variable actual y además en la comprobación de la validez de la solución parcial se incluiría la comprobación de que todas las variables futuras tienen aún valores. El proceso de Forward Checking se puede ver cómo aplicar un simple paso de arco-consistencia sobre cada restricción que involucra la variable actual con una variable futura después de cada asignación de variable. El pseudocódigo de algoritmo Forward Checking es:

Inicio
Seleccionar x_i
Instanciar $x_i \leftarrow a_i: a_i \in D_i$
Razonar hacia adelante (forward-check)
/* Eliminar de los dominios de las variables $(x_{i+1}, \ldots, x_n)$ aún no instanciadas,
aquellos valores inconsisentes con respecto a la instanciación (x_i, a_i), de acuerdo
al conjunto de restricciones */
Si quedan valores posibles en los dominios de todas las variables por instanciar Entonces
Si $i < n$ Entonces
$i = i + 1$
Volver al inicio
Fin Si
Si $i = n$ Entonces
Salir con la solución
Fin Si
Fin Si
Si existe una variable por instanciar, sin valores posibles en su dominio Entonces
Retractar los efectos de la asignación $x_i \leftarrow a_i$
Si quedan valores por intentar en D_i
Volver a instanciar
Sino
Si $i > 1$ Entonces
$i = i - 1$
Volver al inicio
Fin Si
Si $i = 1$ Entonces

Salir con la solución

Fin Si

Fin Si

Fin Si

Fin

Figura N° 3.2.13: Algoritmo Backtracking Cronológico

Ejemplo: El siguiente problema ilustra el comportamiento de Forward Checking:

- 4 familias A, B, C y D viven en casas próximas cuyo s números son: 1, 2, 3 y 4.
- D vive en una casa con menor número que B.
- B vive junto a A en una casa con mayor número.
- Hay al menos una casa entre B y C.
- D no vive en una casa cuyo número es 2.
- C no vive en una casa cuyo número es 4.

¿Qué familia vive en cada casa?

En la siguiente figura, es posible apreciar la representación del grafo de restricciones generado por el problema. Los dominios de cada variable han sido representados según las restricciones descritas arriba.

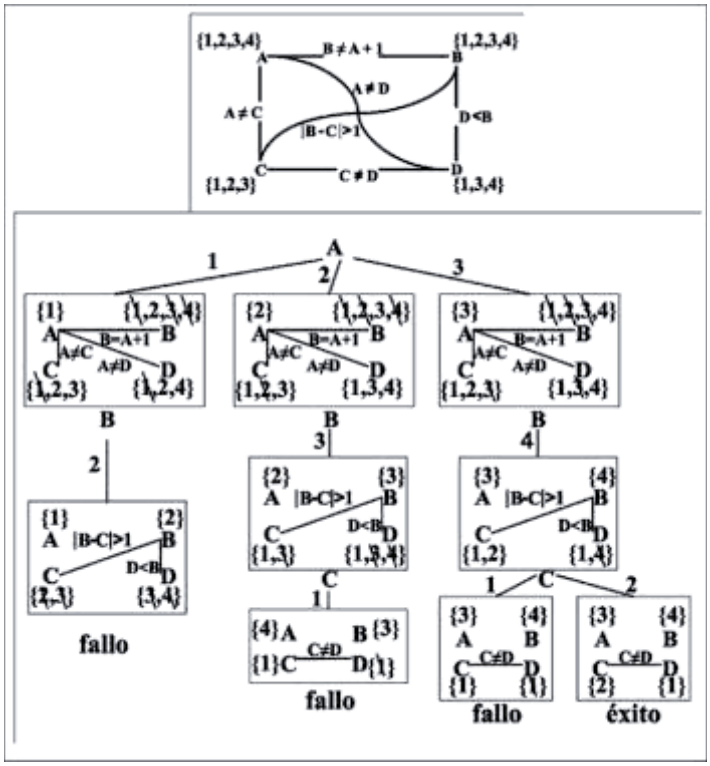


Figura N° 3.2.14: Ejemplo de aplicación Forward Checking

Como se puede apreciar en la Figura, Forward Checking va eliminando los valores de los dominios de las futuras variables luego de realizar una instanciación mediante la propagación de las restricciones, de ésta manera es posible reducir las vueltas atrás (backtracking) y la cantidad de asignaciones requeridas para encontrar la solución, esto, en comparación con Backtracking cronológico.

Minimal Forward Checking

Este método "supone" una mejora del Forward Checkin. En lugar de llevar toda la comprobación de consistencia hacia adelante, Minimal Forward Checking lo realiza cuando es absolutamente necesario.

Para implementar esta mejora en la comprobación, Minimal Forward Checking sólo comprueba si la asignación actual causa una limpieza de dominios. Para hacer esto es suficiente comprobar la asignación actual con los v alores de cada variable futura hasta que se encuentra una que es consistente.

Después, si el algoritmo ha retrocedido, vuelve atrás y lleva a cabo las comprobaciones 'perdidas'. Claramente, Minimal Forward Checking siempre lleva a cabo a lo sumo el mismo número de comprobaciones que Forward Checking. Resultados experimentales han demostrado que la ganancia no supera el 10% [12].

3.2.1.4 Estrategias de Enumeración

La eficiencia de la resolución puede ser mejorada seleccionando un correcto orden de las variables, valores incluso de las restricciones.

En conjunto, una heurística de selección de variable y una heurística de selección de valor, constituyen lo que se conoce como una Estrategia de Enumeración [13]. En el ámbito de la Programación con Restricciones, estas estrategias son cruciales en el rendimiento del proceso de resolución, donde una elección adecuada puede reducir considerablemente el costo computacional de buscar una solución a un CSP.

Las heurísticas de ordenación de variables tratan de seleccionar lo antes posible las variables que más restringen a las demás. Según esta heurística, asignar lo antes posible las variables más restringidas permite identificar las situaciones sin salida antes, logrando reducir el número de vueltas atrás (backtracking). La ordenación de variables puede ser dividida estática y dinámica.

Las heurísticas de ordenación de variables estáticas generan un orden de las variables antes de iniciar la búsqueda, basado en información global derivada del grafo de restricciones inicial.

Las heurísticas de ordenación de variables dinámicas pueden cambiar el orden de las variables dinámicamente basándose en información local que se genera durante la búsqueda.

Ordenación de variables estáticas

Estas heurísticas se basan en la información global que se deriva de la topología del grafo de restricciones original que representa el CSP.

Las heurísticas de ordenación de variables estática generan un orden fijo de las variables antes de iniciar la búsqueda, basado en información global derivada de la estructura de la red de restricciones original que representa el CSP. Antes de listar las heurísticas, primero es necesario comprender algunos conceptos que estas heurísticas manejan.

Una red de restricciones ordenada es una red cuyos nodos han sido ordenados linealmente. En la Figura 10, se observa los diferentes órdenes lineales (de arriba a abajo) posibles para una red inicial de tres variables: $(a,b,c),(a,c,b),(b,a,c)$, etc.

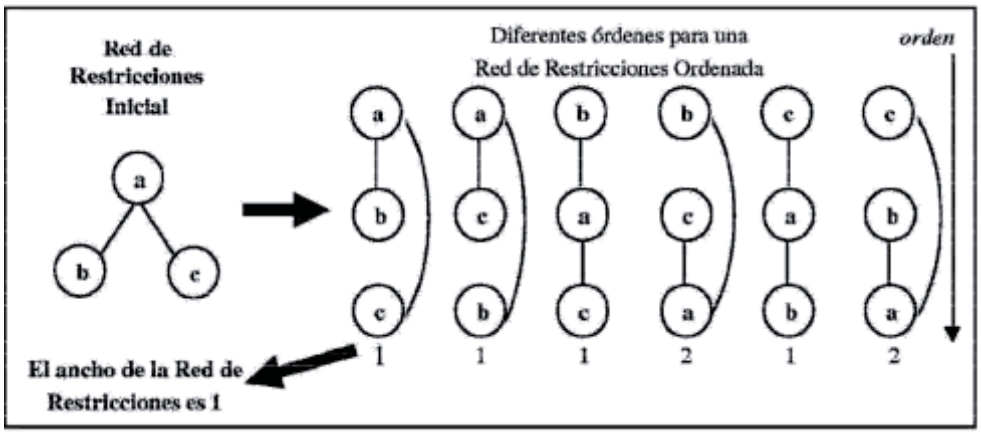


Figura N° 3.2.11: Red de restricciones ordenadas

El ancho de una variable en una red de restricciones ordenada es el número de arcos que restringen a esa variable con las variables superiores, denominadas padres, en el orden lineal de las variables establecido. En la figura de arriba es posible observar los distintos anchos generados luego de que la red fuese ordenada.

El ancho de cada red de restricciones ordenada es el máximo ancho de todas sus variables. De esta forma, el ancho de una red de restricciones es el mínimo ancho de todas sus posibles ordenaciones lineales.

Si una red de restricciones es fuertemente k-consistente, siendo k mayor que el ancho de la red, entonces existe un orden de asignación de variables que no provocaría vuelta atrás en el proceso de búsqueda.

Entre las heurísticas más utilizadas podemos mencionar:

- **Heurística Mínimum Width:** Impone en primer lugar un orden total sobre las variables, de forma que el orden tiene la mínima anchura, y entonces selecciona las variables en base a ese orden. La anchura de la variable x es el número de variables que están antes de x, de acuerdo a un orden dado, y que son adyacentes a x. La anchura de un orden es la máxima anchura de todas así variables bajo ese orden. La anchura de un grafo de restricciones es la anchura mínima de todos los posibles órdenes. Después de calcular la anchura de un grafo de restricciones, las variables se ordenan desde la última hasta la primera en anchura decreciente. Esto significa que las variables que están al principio de la ordenación son las más restringidas y las variables que están al final de la ordenación son las menos restringidas. Asignando las variables más restringidas al principio, las situaciones sin salida se pueden identificar antes y además se reduce el número de vueltas atrás.

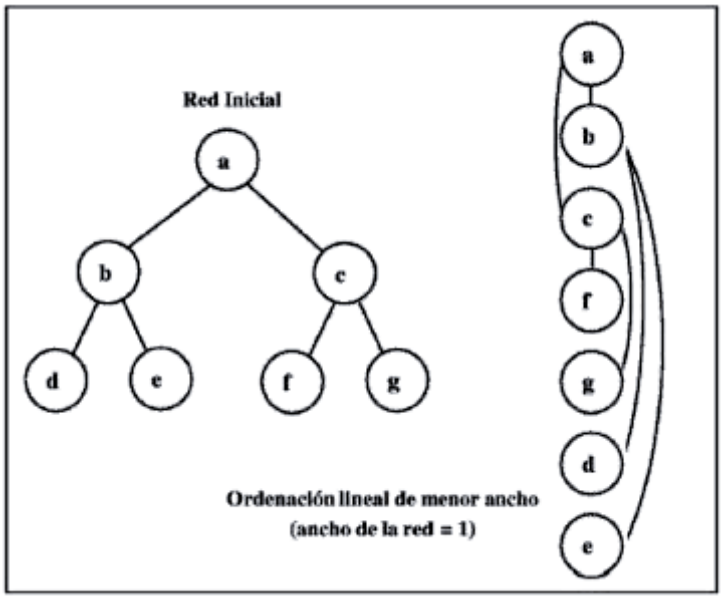


Figura N° 3.2.12: Aplicación de la Heurística de Min Width para red de ancho 1

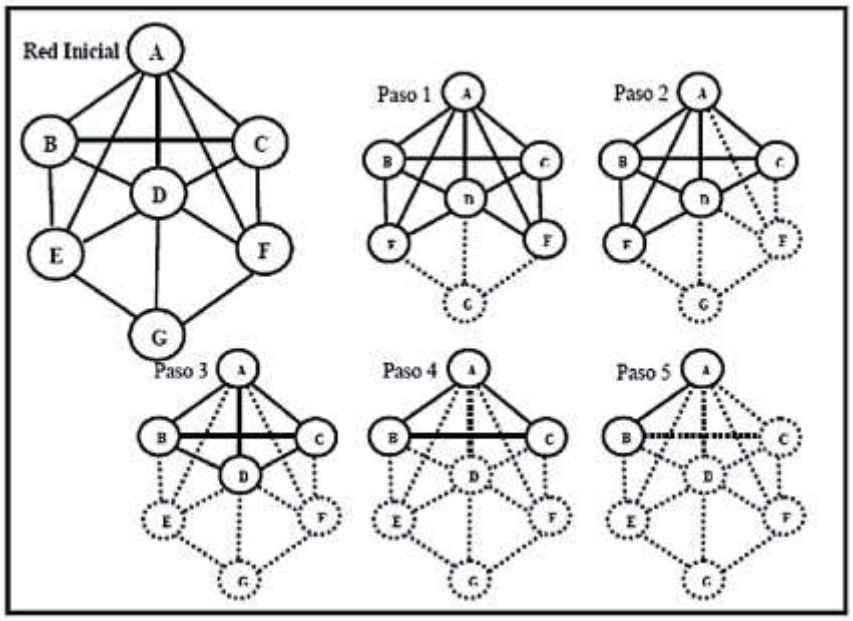


Figura N° 3.2.13: Aplicación de la Heurística de Mínimum Width de ancho 3

- **Heurística Maximum Degree:** Ordena las variables en un orden decreciente de su grado en el grafo de restricciones. El grado de un nodo se define como el número de nodos que son adyacentes a el. Esta heurística también tiene como objetivo encontrar un orden de anchura mínima, aunque no lo garantiza.

- **Heurística Maximum Cardinality:** Selecciona la primera variable arbitrariamente y después en cada paso, selecciona la variable que es adyacente al conjunto más grande de las variables ya seleccionadas.

Ordenación de variables dinámicas

El problema de los algoritmos de ordenación estáticos es que no tienen en cuenta los cambios en los dominios de las variables causados por la propagación de las restricciones durante la búsqueda. Esto es porque estas heurísticas utilizan algoritmos de comprobación hacia atrás donde no se lleva a cabo la propagación de restricciones y fijan el orden de selección antes de iniciar la búsqueda.

La heurística de ordenación de variables dinámicas más común se basa en el principio de primer fallo que sugiere que para tener éxito se debe intentar primero donde sea más probable que falle. De esta manera las situaciones sin salida pueden identificarse antes y además se ahorra espacio de búsqueda. De acuerdo con el principio de primer fallo, en cada paso, se selecciona la variable más restringida, la heurística que refleja es Heurística Mínimum Remaining Values. Entre las heurísticas de ordenación de variables más utilizadas están:

- **Heurística Mínimum Remaining Values (MRV):** Esta heurística selecciona, en cada paso, la variable con el dominio de instanciación más pequeño. Ésta heurística basa en la intuición de que si una variable tiene p ocos valores, entonces es más difícil encontrar un valor consistente. Cuando se utiliza MRV junto con algoritmos de comprobación hacia atrás (4.3.1.2 Backtracking Cronológico), equivale a una heurística estática que ordena las variables de forma ascendente con la talla del dominio antes de llevar a cabo la búsqueda. Cuando MRV se utiliza en conjunción con algoritmos Forward Checking (4.3.3.1.1 Forward Checking), la ordenación se vuelve dinámica, ya que los valores de las futuras variables se pueden podar después de cada asignación de variables. En cada etapa de la búsqueda, la próxima variable a asignarse es la variable con el dominio más pequeño.

- **Heurística Máximum Cardinality**: Esta heurística selecciona la primera variable arbitrariamente y después en cada paso, selecciona la variable que está relacionada con el mayor número de variables ya instanciadas. La intuición es que resulta la más restringida, al estar relacionada con el mayor número de variables ya instanciadas. Esta heurística resulta similar a la Heurística Maximum Degree definida anteriormente, pero en este caso es dinámico. Como una variación de la misma, podría seleccionarse, dinámicamente en cada paso, al variable de máximo grado, sin contabilizar las variables ya instanciadas.

Heurísticas de ordenación de Valores

La idea básica que hay detrás de las heurísticas de ordenación de valores es seleccionar el valor de la variable actual que más probabilidad tenga de llevar a una solución. La mayoría de las heurísticas propuestas tratan de seleccionar el valor menos restringido de la variable actual, es decir, el valor que menos reduce el número de valores útiles para las futuras variables. Algunas heurísticas de ordenación de valor:

- **Min-conflicts:** Es una de las heurísticas de ordenación de valores más conocidas. Esta heurística ordena los valores de acuerdo a los conflictos en que están involucrados con las variables no instanciadas. Esta heurística asocia a cada valor a de la variable actual, el número total de valores en los dominios de las futuras variables adyacentes que son incompatibles con a. El valor seleccionado es el asociado a la suma más baja. Cada valor a de la variable x_i se asocia con el número total de tuplas que son incompatibles con a en las restricciones en las que están involucrada la variable x_i. De nuevo se selecciona el valor con la menor suma. De nuevo los porcentajes se añaden para todas las variables futuras y se selecciona el valor más bajo que se obtiene en todas las sumas.

- **Max-domain-size:** Alternativamente a la anterior, esta heurística selecciona el valor de la variable actual que deja el máximo dominio en las variables futuras. Ésta heurística especifica una manera de romper empates en el método anterior, en el caso de que existan varios valores de la variable actual que dejen el mismo máximo

dominio en las variables futuras. Entonces, se basa en el número de futuras variables que tienen el dominio de mayor tamaño.

Ejemplo: Si un valor a_i deja cinco variables futuras con dominios de seis elementos (y el resto con dominios mayores de seis), y otro valor a_j deja siete variables futuras también con dominios de seis elementos, en este caso se selecciona el valor a_i.

- **Point-domain-size:** Asigna un peso (unidades) a cada valor de la variable actual dependiendo del número de variables futuras que se quedan con ciertos tamaños de dominios.

  Ejemplo: Para cada variable futura que se queda con un dominio de tamaño 1 debido al valor a_i, se añaden 4 unidades al peso de a_i, para cada variable futura que se queda con un dominio de talla 2 se añaden 7 unidades, etc. De esta manera se selecciona el valor con el menor peso.

- **Promise:** Ésta heurística trata de seleccionar el valor que deja un mayor número de soluciones posibles después de que este valor se haya asignado a la variable actual. Para cada valor de la variable se cuenta el número de valores que soporta en cada variable adyacente futura, y toma el producto de las cantidades contadas. Este producto se llama la promesa de un valor. De esta manera se selecciona el valor con la máxima promesa.

3.3 Metaheurísticas

En Inteligencia Artificial (IA) se emplea el calificativo heurístico, en un sentido muy genérico, para aplicarlo a todos aquellos aspectos que tienen que ver con el empleo de conocimiento en la realización dinámica de tareas. Se habla de heurística para referirse a una técnica, método o procedimiento inteligente de realizar una tarea que no es producto de un riguroso análisis formal, sino de conocimiento experto sobre la tarea. En especial, se usa el término heurístico para referirse a un procedimiento que trata de aportar soluciones a un problema con un buen rendimiento, en lo referente a la calidad de las soluciones y a los recursos empleados. En la resolución de problemas específicos han surgido procedimientos heurísticos exitosos, de los que se ha tratado de extraer lo que es esencial en su éxito para aplicarlo a otros problemas o en contextos más extensos. Como ha ocurrido claramente en diversos campos de la IA, en especial con los sistemas expertos, esta línea de investigación ha contribuido al desarrollo científico del campo de las heurísticas y a extender la aplicación de sus resultados. De esta forma se han obtenido, tanto técnicas y recursos computacionales específicos, como estrategias de diseño generales para procedimientos heurísticos de resolución de problemas. Estas estrategias generales para construir algoritmos, que quedan por encima de las heurísticas, y van algo más allá, se denominan metaheurísticas. Las metaheurísticas pueden integrarse como un sistema experto para facilitar su uso genérico a la vez que mejorar su rendimiento [14].

3.4 Algoritmos Genéticos

Los Algoritmos Genéticos son métodos adaptativos, generalmente usados en problemas de búsqueda y optimización de parámetros, basados en la reproducción y en el principio supervivencia del más apto.

Más formalmente, y siguiendo la definición dada por Goldberg, ``Los Algoritmos Genéticos son algoritmos de búsqueda basados en la mecánica de selección natural y de la genética natural. Combinan la supervivencia del más apto entre estructuras de secuencias con un intercambio de información estructurado, aunque aleatorio, para constituir así un algoritmo de búsqueda que tenga algo de las genialidades de las búsquedas humanas" [15].
Para alcanzar la solución a un problema se parte de un conjunto inicial de individuos, llamado población, generado de manera aleatoria. Cada uno de estos individuos representa una posible solución al problema. Estos individuos evolucionarán tomando como base los esquemas propuestos por Darwin sobre la selección natural, y se adaptarán en mayor medida tras el paso de cada generación a la solución requerida [16].

Cualquier solución potencial a un problema puede ser presentada dando valores a una serie de parámetros. El conjunto de todos los parámetros (genes en la terminología de Algoritmos Genéticos) se codifica en una cadena de valores denominada cromosoma.

El conjunto de los parámetros representado por un cromosoma particular recibe el nombre de genotipo. El genotipo contiene la información necesaria para la construcción del organismo, es decir, la solución real al problema, denominada fenotipo. Por ejemplo, en términos biológicos, la información genética contenida en el ADN de un individuo sería el genotipo, mientras que la expresión de ese ADN (el propio individuo) sería el fenotipo.

Desde los primeros trabajos de John Holland la codificación suele hacerse mediante valores binarios. Se asigna un determinado número de bits a cada parámetro y se realiza una discretización de la variable representada por cada gen. El número de bits asignados dependerá del grado de ajuste que se desee alcanzar. Evidentemente no todos los parámetros tienen porque estar codificados con el mismo número de bits. Cada uno de los bits pertenecientes a un gen suele recibir el nombre de alelo.

La siguiente figura muestra un ejemplo de un individuo binario que codifica 3 parámetros.

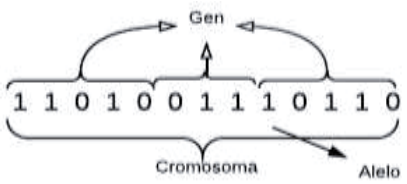


Figura N° 3.2.1: Individuo Genético Binario

Sin embargo, también pueden existir representaciones que codifiquen directamente cada parámetro con un valor entero, real o en punto flotante. A pesar de que se acusa a estas representaciones de degradar el paralelismo implícito de las representaciones binarias, permiten el desarrollo de operadores genéticos más específicos al campo de aplicación del Algoritmo Genético.

Los Algoritmos Genéticos trabajan sobre una población de individuos. Cada uno de ellos representa una posible solución al problema que se desea resolver. Todo individuo tiene asociado un ajuste de acuerdo a la aptitud con respecto al problema de la solución que representa (en la naturaleza el equivalente sería una medida de la eficiencia del individuo en la lucha por los recursos).

El funcionamiento genérico de un Algoritmo Genético puede apreciarse en el siguiente pseudocódigo:

```
Inicio
    Generar una población inicial
    Evaluar el fitness de los individuos de la población
        Mientras
            Seleccionar individuos padres desde la población
            Aplicar los operadores genéticos para generar nuevos individuos
            Evaluar el fitness de los individuos hijos
            Reemplazar algunos o todos los individuos de la población por los hijos
        Hasta Condición de Término
Fin
```

Figura N° 3.2.12: Algoritmo Genético

Una generación se obtiene a partir de la anterior por medio de los operadores de reproducción. Existen 2 tipos:

- **Cruce**: Se trata de forma de calcular el genoma del nuevo individuo en función del genoma del padre y de la madre. Se genera una descendencia a partir del mismo número de individuos (generalmente 2) de la generación anterior.

- **Copia**: Se trata de una reproducción que no involucra al genoma padre ni al genoma madre. Un determinado número de individuos pasa sin sufrir ninguna variación directamente a la siguiente generación.

Una vez generados los nuevos individuos se realiza la mutación con una probabilidad Pm. La probabilidad de mutación suele ser muy baja, por lo general entre el 0.5% y el 2%.

El proceso acaba cuando se alcanza alguno de los criterios de parada fijados. Los más usuales suelen ser:

- Los mejores individuos de la población representan soluciones suficientemente buenas para el problema que se desea resolver.

- La población ha convergido. Un gen ha convergido cuando el 95% de la población tiene el mismo valor para él, en el caso de trabajar con codificaciones binarias, o valores dentro de un rango especificado, en el caso de trabajar con otro tipo de codificaciones. Una vez que todos los genes alcanzan la convergencia se dice que la población ha convergido. Cuando esto ocurre la media de bondad de la población se aproxima a la bondad del mejor individuo.

- Se ha alcanzado el número de generaciones máximo especificado.

Sobre este algoritmo inicialmente propuesto por Holland se han definido numerosas variantes. Quizás una de las más extendidas consiste en prescindir de la población temporal de manera que los operadores genéticos de cruce y mutación se aplican directamente sobre la población genética. Con esta variante el proceso de cruces varía ligeramente. Ahora no basta, en el caso de que el cruce se produzca, con insertar directamente la descendencia en la población. Puesto que el número de individuos de la población se ha de mantener constante, antes de insertar la descendencia en la población se le ha de hacer sitio. Existen para ello diversas opciones:

- Remplazo de padres: para hacer hueco a la descendencia en la población se eliminan de ella a los padres.

- Remplazo de individuos similares: cada uno de los individuos de la descendencia remplazará a un individuo de la población con un valor de aptitud similar al suyo. Para escoger este individuo se obtiene la posición en la que se debería insertar el nuevo individuo para mantener ordenada la población y se escoge para insertarlo una posición al azar de su población.

- Remplazo de los peores individuos: los individuos que se eliminarán de la población para dejar paso a la descendencia se seleccionarán aleatoriamente de entre los peores individuos de la población. Por lo general se consideran individuos pertenecientes al último 10%.

- Remplazo aleatorio: los individuos eliminados se seleccionan al azar.

Evidentemente trabajando con una única población no se puede decir que se pase a la siguiente generación cuando se llene la población, pues siempre está llena. En este caso el paso a la siguiente generación se producirá una vez que se hayan alcanzado cierto número de cruces y mutaciones. Este número dependerá de la tasa de cruces y mutaciones

especificadas por el usuario y del tamaño de la población. Así con una tasa de cruces del 90%, una tasa de mutaciones del 0.02% y trabajando con 100 individuos se pasará a la siguiente generación cuando se alcanzasen 45 cruces (cada cruce genera 2 individuos con lo que se habrían insertado en la población 90 individuos, esto es el 90%) o 2 mutaciones.

Para el paso de una generación a la siguiente se aplican una serie de operadores genéticos. Los más empleados son los operadores de selección, cruce, copia y mutación. En el caso de no trabajar con una población intermedia temporal también cobran relevancia los algoritmos de remplazo [17].

3.4.1 Selección

Los algoritmos de selección serán los encargados de escoger qué individuos van a disponer de oportunidades de reproducirse y cuáles no.

Puesto que se trata de imitar lo que ocurre en la naturaleza, se ha de otorgar un mayor número de oportunidades de reproducción a los individuos más aptos. Por lo tanto la selección de un individuo estará relacionada con su valor de aptitud. No se debe sin embargo eliminar por completo las opciones de reproducción de los individuos menos aptos, pues en pocas generaciones la población se volvería homogénea.

Una opción bastante común consiste en seleccionar el primero de los individuos participantes en el cruce mediante alguno de los métodos expuestos a continuación y el segundo de manera aleatoria.

Selección por ruleta

Propuesto por De Jong, es posiblemente el método más utilizado desde los orígenes de los Algoritmos Genéticos [18]. A cada uno de los individuos de la población se le asigna una parte proporcional a su ajuste de una ruleta, de tal forma que la suma de todos los porcentajes sea uno. Los mejores individuos recibirán una porción de la ruleta mayor que la recibida por los peores. Generalmente la población está ordenada en base al ajuste por lo que las porciones más grandes se encuentran al inicio de la ruleta. Para seleccionar un individuo basta con generar un número aleatorio del intervalo [0,1] y devolver el individuo situado en esa posición de la ruleta. Esta posición se suele obtener recorriendo los individuos de la población y acumulando sus proporciones de ruleta hasta que la suma exceda el valor obtenido.

Es un método muy sencillo, pero ineficiente a medida que aumenta el tamaño de la población. Presenta además el inconveniente de que el peor individuo puede ser seleccionado más de una vez. En mucha bibliografía se suele referenciar a este método con el nombre de Selección de Montecarlo.

Selección por torneo

La idea principal de este método consiste en realizar la selección en base a comparaciones directas entre individuos. Existen dos versiones de selección mediante torneo:

- Determinística
- Probabilística

En la versión determinística se selecciona al azar un número p de individuos. De entre los individuos seleccionados se selecciona el más apto para pasarlo a la siguiente generación.

La versión probabilística únicamente se diferencia en el paso de selección del ganador del torneo. En vez de escoger siempre el mejor se genera un número aleatorio del intervalo [0,1], si es mayor que un parámetro p (fijado para todo el proceso evolutivo) se escoge el individuo más alto y en caso contrario el menos apto.

Variando el número de individuos que participan en cada torneo se puede modificar la presión de selección. Cuando participan muchos individuos en cada torneo, la presión de selección es elevada y los peores individuos apenas tienen oportunidades de reproducción. Un caso particular es el elitismo global. Se trata de un torneo en el que participan todos los individuos de la población con lo cual la selección se vuelve totalmente determinística. Cuando el tamaño del torneo es reducido, la presión de selección disminuye y los peores individuos tienen más oportunidades de ser seleccionados.

Elegir uno u otro método de selección determinará la estrategia de búsqueda del Algoritmo Genético. Si se opta por un método con una alta presión de selección se centra la búsqueda de las soluciones en un entorno próximo a las mejores soluciones actuales. Por el contrario, optando por una presión de selección menor se deja el camino abierto para la exploración de nuevas regiones del espacio de búsqueda.

Existen muchos otros algoritmos de selección. Unos buscan mejorar la eficiencia computacional, otros el número de veces que los mejores o peores individuos pueden ser seleccionados. Algunos de estos algoritmos son muestreo determinístico, escalamiento sigma, selección por jerarquías, estado uniforme, sobrante estocástico, brecha generacional [19].

3.4.2 Cruce

Una vez seleccionados los individuos, éstos son recombinados para producir la descendencia que se insertará en la siguiente generación. Tal y como se ha indicado anteriormente el cruce es una estrategia de reproducción sexual.

Su importancia para la transición entre generaciones es elevada puesto que las tasas de cruce con las que se suele trabajar rondan el 90%.

Los diferentes métodos de cruce podrán operar de dos formas diferentes. Si se opta por una estrategia destructiva los descendientes se insertarán en la población temporal aunque sus padres tengan mejor ajuste (trabajando con una única población esta comparación se realizará con los individuos a remplazar). Por el contrario utilizando una estrategia no destructiva la descendencia pasará a la siguiente generación únicamente si supera la bondad del ajuste de los padres (o de los individuos a remplazar). La idea principal del cruce se basa en que, si se toman dos individuos correctamente adaptados al medio y se obtiene una descendencia que comparta genes de ambos, existe la posibilidad de que los genes heredados sean precisamente los causantes de la bondad de los padres. Al compartir las características buenas de dos individuos, la descendencia, o al menos parte de ella, debería tener una bondad mayor que cada uno de los padres por separado. Si el cruce no agrupa las mejores características en uno de los hijos y la descendencia tiene un peor ajuste que los padres no significa que se esté dando un paso atrás. Optando por una estrategia de cruce no destructiva garantizamos que pasen a la siguiente generación los mejores individuos. Si, aún con un ajuste peor, se opta por insertar a la descendencia, y puesto que los genes de los padres continuarán en la población - aunque dispersos y posiblemente levemente modificados por la mutación - en posteriores cruces se podrán volver a obtener estos padres, recuperando así la bondad previamente pérdida [20].

Existen multitud de algoritmos de cruce. Sin embargo los más empleados son los que se detallarán a continuación:

- Cruce de 1 punto
- Cruce de 2 puntos
- Cruce uniforme o por máscara

Cruce de 1 punto

Es la más sencilla de las técnicas de cruce. Una vez seleccionados dos individuos se cortan sus cromosomas por un punto seleccionado aleatoriamente para generar dos segmentos diferenciados en cada uno de ellos: la cabeza y la cola. Se intercambian las colas entre los dos individuos para generar los nuevos descendientes. De esta manera ambos descendientes heredan información genética de los padres, tal y como puede verse en la siguiente figura:

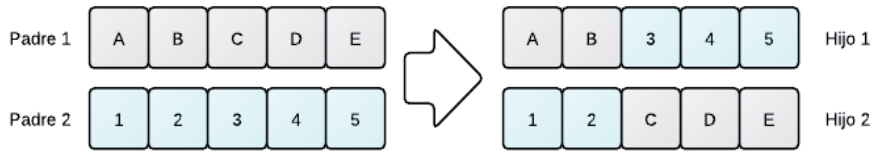


Figura N° 3.4.2.1: Cruce de 1 Punto

Cruce de 2 puntos

Se trata de una generalización del cruce de 1 punto. En vez de cortar por un único punto los cromosomas de los padres como en el caso anterior se realizan dos cortes. Deberá tenerse en cuenta que ninguno de estos puntos de corte coincida con el extremo de los cromosomas para garantizar que se originen tres segmentos. Para generar la descendencia se escoge el segmento central de uno de los padres y los segmentos laterales del otro padre como se aprecia en la siguiente figura:

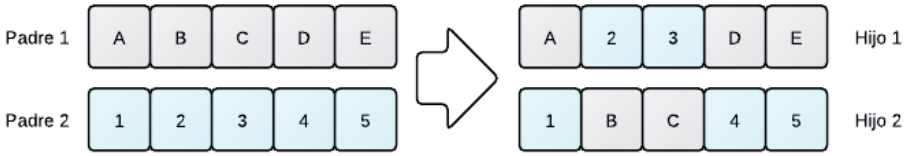


Figura N° 3.4.2.2: Cruce de 2 Puntos

Generalizando se pueden añadir más puntos de cruce dando lugar a algoritmos de cruce multipunto. Sin embargo existen estudios que desaprueban esta técnica. Aunque se admite que el cruce de 2 puntos aporta una sustancial mejora con respecto al cruce de un solo punto, el hecho de añadir un mayor número de puntos de cruce reduce el rendimiento del Algoritmo Genético. El problema principal de añadir nuevos puntos de cruce radica en que es más fácil que los segmentos originados sean corrompibles, es decir, que por separado quizás pierdan las características de bondad que poseían conjuntamente. Sin embargo no todo son desventajas y añadiendo más puntos de cruce se consigue que el espacio de búsqueda del problema sea explorado más a fondo.

Cruce Uniforme o por Máscara

El cruce uniforme es una técnica completamente diferente de las vistas hasta el momento. Cada gen de la descendencia tiene las mismas probabilidades de pertenecer a uno u otro padre.

Aunque se puede implementar de muy diversas formas, la técnica implica la generación de una máscara de cruce con valores binarios. Si en una de las posiciones de la máscara hay un 1, el gen situado en esa posición en uno de los descendientes se copia del primer padre. Si por el contrario hay un 0 el gen se copia del segundo padre. Para producir el segundo descendiente se intercambian los papeles de los padres, o bien se intercambia la interpretación de los unos y los ceros de la máscara de cruce.

En la siguiente figura se puede apreciar un ejemplo de cruce por Máscara o Cruce Uniforme

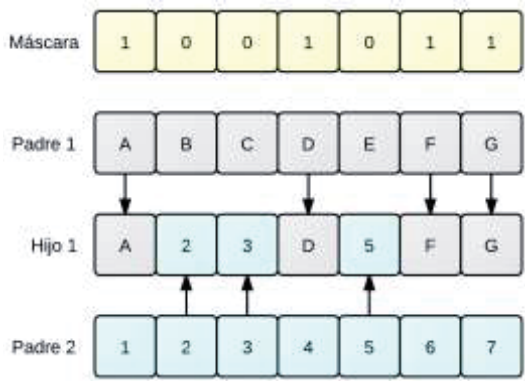


Figura N° 3.4.2.3: Cruce Uniforme

Algoritmos de Remplazo

Cuando en vez de trabajar con una población temporal se hace con una única población, sobre la que se realizan las selecciones e inserciones, deberá tenerse en cuenta que para insertar un nuevo individuo deberá de eliminarse previamente otro de la población. Existen diferentes métodos de remplazo:

- Aleatorio: el nuevo individuo se inserta en un lugar cualquiera de la población.

- Remplazo de padres: se obtiene espacio para la nueva descendencia liberando el espacio ocupado por los padres.
- Remplazo de similares: una vez obtenido el ajuste de la descendencia se selecciona un grupo de individuos (entre seis y diez) de la población con un ajuste similar. Se remplazan aleatoriamente los que sean necesarios.

- Remplazo de los peores: de entre un porcentaje de los peores individuos de la población se seleccionan aleatoriamente los necesarios para dejar sitio a la descendencia.

3.4.3 Copia

La copia es la otra estrategia reproductiva para la obtención de una nueva generación a partir de la anterior. A diferencia del cruce, se trata de una estrategia de reproducción asexual. Consiste simplemente en la copia de un individuo en la nueva generación.

El porcentaje de copias de una generación a la siguiente es relativamente reducido, pues en caso contrario se corre el riesgo de una convergencia prematura de la población hacia ese individuo. De esta manera el tamaño efectivo de la población se reduciría notablemente y la búsqueda en el espacio del problema se focalizaría en el entorno de ese individuo.

Lo que generalmente se suele hacer es seleccionar dos individuos para el cruce, y si éste finalmente no tiene lugar, se insertan en la siguiente generación los individuos seleccionados.

3.4.4 Mutación

La mutación de un individuo provoca que alguno de sus genes, generalmente uno sólo, varíe su valor de forma aleatoria. Aunque se pueden seleccionar los individuos directamente de la población actual y mutarlos antes de introducirlos en la nueva población, la mutación se suele utilizar de manera conjunta con el operador de cruce. Primeramente se seleccionan dos individuos de la población para realizar el cruce. Si el cruce tiene éxito entonces uno de los descendientes, o ambos, se muta con cierta probabilidad Pm. Se imita de esta manera el comportamiento que se da en la naturaleza, pues cuando se genera la descendencia siempre se produce algún tipo de error, por lo general sin mayor trascendencia, en el paso de la carga genética de padres a hijos.
La probabilidad de mutación es muy baja, generalmente menor al 1%. Esto se debe sobre todo a que los individuos suelen tener un valor de aptitud menor después de mutados. Sin embargo se realizan mutaciones para garantizar que ningún punto del espacio de búsqueda tenga una probabilidad nula de ser examinado.
Tal y como se ha comentado, la mutación más usual es el remplazo aleatorio. Este consiste en variar aleatoriamente un gen de un cromosoma. Si se trabaja con codificaciones binarias consistirá simplemente en negar un bit. También es posible realizar la mutación intercambiando los valores de dos alelos del cromosoma.

3.4.5 Evaluación

Para el correcto funcionamiento de un Algoritmo Genético se debe de poseer un método que indique si los individuos de la población representan o no buenas soluciones al problema planteado. Por lo tanto para cada tipo de problema que se desee resolver deberá derivarse un nuevo método, al igual que ocurrirá con la propia codificación de los individuos.

De esto se encarga la función de evaluación, que establece una medida numérica de la bondad de una solución. Esta medida recibe el nombre de ajuste. En la naturaleza el ajuste (o adecuación) de un individuo puede considerarse como la probabilidad de que ese individuo sobreviva hasta la edad de reproducción y se reproduzca. Esta

probabilidad deberá estar ponderada con el número de descendientes. Evidentemente no es lo mismo una probabilidad de reproducción del 25% en una población de un par de cientos de individuos que esa misma probabilidad en una población de varios millones.

En el mundo de los Algoritmos Genéticos se empleará esta medición para controlar la aplicación de los operadores genéticos. Es decir, permitirá controlar el número de selecciones, cruces, copias y mutaciones llevadas a cabo.
La aproximación más común consiste en crear explícitamente una medida de ajuste para cada individuo de la población. A cada uno de los individuos se les asigna un valor de ajuste escalar por medio de un procedimiento de evaluación bien definido. Tal y como se ha comentado, este procedimiento de evaluación será específico del dominio del problema en el que se aplica el Algoritmo Genético. También puede calcularse el ajuste mediante una manera 'co-evolutiva'. Por ejemplo, el ajuste de una estrategia de juego se determina aplicando esa estrategia contra la población entera (o en su defecto una muestra) de estrategias de oposición.

3.4.6 Criterios de convergencia

El criterio de convergencia permite definir un punto de término de la ejecución de la Metaheurística en búsqueda de soluciones, los criterios de convergencia más utilizados son los siguientes:

Criterio de convergencia de identidad, este criterio consiste en detener al algoritmo genético cuando un determinado porcentaje de los cromosomas de la población representan a la misma solución. Los operadores del algoritmo genético tienen a preservar y difundir el material genético de los cromosomas más aptos, por lo que es de esperar que luego de un gran número de generaciones, alguna solución con gran valor de aptitud se imponga y domine la población

Convergencia de aptitud, puede suceder que existan soluciones equivalentes o casi equivalentes a un problema, que obtengan valores de aptitud similares. En este caso, es probable que no haya una solución que se imponga en la población (y el criterio de terminación por convergencia de identidad nunca se cumpla). Este criterio no espera a que la población se componga mayoritariamente de una sola solución, sino que finaliza la ejecución del algoritmo cuando los valores de aptitud de un determinado porcentaje de las soluciones son iguales, o difieren en un pequeño porcentaje. Por ejemplo, cuando el 90% de las soluciones tenga valores de aptitud que no difieran en más de un 1%.

Criterio de cantidad de generaciones, Los métodos anteriores apuntan a esperar a que la evolución de la población llegue a su fin. Cuando alguno de ellos se cumple, es probable que las soluciones no sigan mejorando mucho más, no importa cuántas generaciones más se ejecuten. Sin embargo, los algoritmos genéticos pueden necesitar un número de generaciones muy grande para llegar a la convergencia, dependiendo de las tasas de reproducción y

mutación. Utilizando cualquiera de los dos criterios anteriores no puede estimarse un número máximo de generaciones, ya que esto dependerá no solamente de los parámetros del algoritmo genético sino también del azar. Esto puede ser un problema, sobre todo si se quieren comparar los tiempos de resolución de un problema mediante algoritmos genéticos con otros métodos. El criterio de terminación por cantidad de generaciones consiste simplemente en finalizar la ejecución una vez que ha transcurrido un número determinado de generaciones. Este método permite determinar con precisión los tiempos de ejecución del algoritmo a costa de detener la evolución sin la certeza de que las soluciones no seguirán mejorando.

4 Ejemplo de aplicación del Set Partitioning

Set Partitioning Problem se divide en sub problemas en diversos problemas de optimización combinatorial, como por ejemplo en la programación de las aerolíneas. Una sub tarea de la programación de las aerolíneas, llamado Programación de la tripulación, toma como datos de entrada un conjunto de pares de la tripulación, donde un equipo de emparejamiento es una secuencia de vuelos sin escalas de un único equipo y una pareja empieza y termina en la misma estación base. Tal conjunto de emparejamientos se genera de acuerdo a los vuelos de la aerolínea ofrece a sus clientes, donde un gran número de restricciones tienen que ser tomadas en cuenta, como por ejemplo los contratos de trabajo y horarios de los sindicatos. La selección de los emparejamientos de la tripulación que causan un costo mínimo y garantizar que cada vuelo se cubre exactamente una vez, puede modelado como un Set Partitioning Problem. Por lo general, SPP se resuelven por Solvers de programación lineal entera (ILP) [21].

5 Solución Propuesta

5.1 Explicación General

Dado que se tiene información de comportamiento de los clientes que pertenecen al club de fidelización, se podrá hacer uso de esta para realizar el análisis de los datos a modo de agrupar los comportamiento de los distintos clientes que consumen los servicios entregados por la empresa, si bien ya existen agrupaciones de comportamiento que maneja el área de inteligencia de negocios, estos se solicitarán a fin de tratar de adecuarlos agregándoles o quitándoles características, para utilizarlos en este problema.

Con esta información se generarán nuevas y se mantendrán algunas agrupaciones de clientes, teniendo en cuenta que estas podrían cambiar a medida que pasa el tiempo, con estas se evaluará cuáles de ellas se alejan en mayor medida del comportamiento esperado para los grupos, una vez evaluados esos grupos, se debe generar el tratamiento adecuado para que este comience a acercarse al comportamiento esperado y así los grupos alejados en comportamiento vayan desapareciendo con el tiempo.

El análisis debe tener la inteligencia necesaria para lograr examinar nuevos grupos que se vayan creando con el tiempo para así poder tratarlos de manera dinámica y llevarlos al comportamiento deseado.
Este trabajo se compondrá de dos etapas marcas, las cuales se dividen según el estudio y las técnicas que se utilizarán para ejecutar estos últimos.

La primera etapa del trabajo se enfocará en el tratamiento de los grupos de comportamiento (una vez que estos ya hayan sido aislados), este tratamiento de grupos se hará con respecto a acciones que se ejecutarán sobre estos para que vayan perdurando en el tiempo si es necesario o de lo contrario comiencen a cambiar su comportamiento a través de las acciones que se aplicarán. Para llegar a decidir cuál es la acción más apta para aplicar al grupo se utilizará una de estas técnicas, Programación con restricciones (Constraint Programming) o Metaheurísticas, y en base a esta que entregará las acciones menos costosas a aplicar a los grupos. Si bien la metaheurística podría no entregar las acciones menos costosas puesto que esta técnica es una técnica incompleta y podría arrojar algún mejor resultado local no global a diferencia como lo hace Constraint Programming que explora todo el espectro de soluciones, la decisión de que técnica será utilizada se dará en base a la experimentación y los resultados obtenidas de estas.

Es importante destacar que dentro del universo de clientes existe una agrupación principal llamada categoría que se obtiene a partir de los puntos que estos poseen, los puntos de clientes pueden acumularse por distintos medios, ya sea por la utilización de la tarjeta en alguna máquina de juego, por la compra de alimentos, regalo de puntos, etc.

Las categorías son las siguientes:

Categoría Classic: Categoría básica en la cual el cliente quedará registrado cuando se inscribe en el club de fidelización.
Categoría Silver: Categoría que sigue a la categoría Classic
Categoría Gold: Categoría que sigue a la categoría Silver
Categoría Diamond: Categoría que sigue a la categoría Gold
Categoría Platinum: Categoría que sigue a la categoría Diamond, y corresponden a clientes de alto valor.
Las categorías Classic, Silver y Gold son las que en conjunto alojan la mayor cantidad de clientes.

Los grupos de comportamiento se defienden en base a un conjunto de variable que se miden a los clientes cuando utilizan la tarjeta del club de fidelización, podemos distinguir agrupaciones por:

- Categoría, corresponden a las categorías nombradas anteriormente.

- Unidad de Negocio (Casino en el cual cliente Juega o ha jugado), las unidades actuales que están en funcionamiento se encuentran en las ciudades de Antofagasta, Coquimbo, Viña del Mar, Rinconada de los Andes, Santa Cruz Pucón, Chiloé y Mendoza.

- Rango Etáreo, es el rango de edad en las cuales se agrupan los clientes.

- Frecuencia de Visita, es un índice que indica la frecuencia de visita del cliente a las instalaciones, este índice se puede medir de manera semanal, mensual, trimestral, etc.

- Ganancia promedio por visita, es un indicador que resume el gasto realizado por el cliente en promedio en las visitas hechas a las distintas unidades de negocio, por ende es una ganancia para la organización, de ahí su nombre.

Estas variables nombradas, son sólo algunas en las cuales se basa los distintos grupos de comportamiento utilizados en la actualidad.

Luego de haber explicado la manera en que los grupos de comportamiento de dividen, ahora se aclara como se generan las acciones que potencialmente podrían aplicarse a estos grupos. Las acciones se definen en base a cuatro líneas de servicio, los cuales son:

- Juegos, específicamente corresponde a la entrega de cupones válidos por juego

- Alimentos y Bebidas, en acciones relacionadas a esta línea se entregan cupones por cenas y por bebestibles

- Hotel, por este lado se regalan noches de alojamiento en hoteles de la cadena

- Espectáculos, se entregan cupones de descuento o vales por entradas a espectáculos realizados por la cadena

Dependiendo cada acción, esta tendrá un monto asignado, puesto que estas se trasforman en un consumo para el cliente, ya sea de servicios y de productos. Las acciones asignadas a grupos de clientes están relacionadas directamente al gasto que ellos realizan en las unidades de negocio, es decir, a clientes con mayor gasto, por ende, mejor categoría, acciones de mayor valor.

5.2 Vínculo Acciones Grupos

Como se definió anteriormente existen los grupos de clientes, que contendrán el comportamiento similar de una cierta cantidad de clientes, este listado de agrupaciones lo llamaremos N, y al listado de acciones nombradas anteriormente le llamaremos M, entonces esto se traduce a:

Se tiene N grupos de comportamiento y M acciones que podrán aplicarse a algún grupo, entonces se debe obtener cual es la mejor acción dentro de M que debe aplicarse a un cierto grupo de comportamiento N en base al costo. Cada acción tiene un costo asociado por cliente. Cada grupo tiene un % de inversión, que se traduce al % de inversión que posee el cliente, el porcentaje de inversión indica que % de la ganancia como tope podrá aplicarse al grupo, la ganancia corresponde al valor promedio que gasta el cliente en la visita a las instalaciones.

Pongamos un ejemplo, si un cliente asiste dos veces por semana a las instalaciones y este gasta en promedio $60.000.- y el % de inversión corresponde al 25%, significa que el costo de la acción aplicable a este cliente no puede sobrepasar los $15.000.-

Ahora si el grupo al cual pertenece el cliente posee un % de inversión del 125% esto indica que el costo de la acción aplicable al grupo de cliente por ende aplicable al cliente que en promedio gasta $60.000.- podrá tener un valor de hasta $75.000.-, este comportamiento recae en clientes que tienen una proyección de comportamiento positiva y con los cuales el % de sobre inversión será recuperado en el futuro.

En resumen y explicado de una manera más simple, cada fila de la matriz corresponde a un grupo de clientes donde cada grupo posee un valor de ganancia (GA). Y cada columna es una acción de tipo comercial (regalo de cupones de

juego, noches de hotel entre otros), que posee un costo y puede entregarse a algún grupo. Ahora el punto es que cada grupo o fila posee un porcentaje de inversión de la ganancia, por ende, el valor de la acción comercial a aplicar no debe sobrepasar ese valor (porcentaje de inversión sobre ganancia).

Todo lo detallado anteriormente se grafica en la siguiente matriz.

Matriz de posibles coberturas

GA	PI%		Acción 1	Acción 2	Acción 3	Acción 4	Acción 5	Acción 6	Acción 7	Acción 8	Acción 9	…	Acción N
30.000	10	Grupo 1	1	1	1	0	0	1	0	0	0	1	1
40.000	20	Grupo 2	0	0	0	1	0	0	1	1	0	0	0
50.000	25	Grupo 3	0	1	0	0	0	0	0	0	0	0	0
100.000	30	Grupo 4	1	0	1	1	0	1	0	0	1	0	0
120.000	35	Grupo 5	0	1	1	0	0	0	1	0	0	0	0
150.000	40	Grupo 6	1	0	0	1	0	1	0	0	0	0	1
200.000	45	Grupo 7	0	1	0	0	1	0	0	1	0	0	0
250.000	50	Grupo 8	0	0	0	0	0	0	0	0	1	1	1
300.000	55	Grupo 9	0	1	1	0	0	0	0	0	0	0	1
350.000	60	Grupo 10	1	0	0	0	1	0	0	1	0	1	1
…	…	…											
500.000	80	Grupo M	0	1	0	1	0	0	1	0	1	0	1

Figura N° 4.2.1: Matriz de Restricciones

Y un vector de costos, que indica el valor de la acción a aplicar por cliente (valor unitario)

Acción 1	Acción 2	Acción 3	Acción 4	Acción 5	Acción 6	Acción 7	Acción 8	Acción 9	…	Acción N
Costo 1	Costo 2	Costo 3	Costo 4	Costo 5	Costo 6	Costo 7	Costo 8	Costo 9	…	Costo N

Figura N° 4.2.2: Vector de Costos

Donde además cada Grupo posee dos parámetros:

- Ganancia (GA), indica la ganancia para el casino, en otras palabras lo que desembolsa cada cliente en el juego.

- Porcentaje de inversión (PI): Indica que porcentaje con respecto a la ganancia promedio de cada cliente en el grupo se podrá invertir como costo de la acción, es decir, el costo de esta no debe sobrepasar este valor, ahora el % de la inversión podría ser mayor al 100% esto indica que el valor de costo de la acción puede sobrepasar el valor de la ganancia.

Esto está condicionado a la siguiente restricción:

- Restricción % de inversión: Esta restricción indica que el costo de la acción no debe sobrepasar el valor del porcentaje aplicado a la ganancia promedio, el % de inversión puede ser mayor al 100%, esto indica que el costo de la acción puede sobrepasar el valor de la ganancia.

5.3 Modelo matemático

Según lo expuesto anteriormente y explicitado las distintas variables que se incluirán en el modelo, e indicando que el problema se abarcará como un problema del tipo Set Partitioning el modelo se expresa de la siguiente manera,

Sea:

$A = (a_{ij})$ una matriz binaria (0, 1) de dimensiones m x n, que representa las posibilidades de asignación de Acciones a grupos de Clientes.
$C = (C_j)$ un vector de largo n, que contiene los Costos de las Acciones.
$M = \{1, \dots, m\}$, representa los grupos de Cliente disponibles a los cuales se podrán aplicar las Acciones
$N = \{1, \dots, n\}$, representa las Acciones que se podrán aplicar a los grupos de Clientes.

El valor $c_j (j \in N)$ representa el costo de la columna j, Así, decimos que una Acción $j \in N$ se aplica al grupo de Clientes $i \in M$ si $a_{ij} = 1$

La función Objetivo es la siguiente:

$$Min \sum_{j=1}^{n} c_j x_j \qquad (3)$$

Lo que se traduce en minimizar los costos de las Acciones que se asignan a los grupos de Clientes.

Sujeto a:

$$\sum_{j=1}^{n} a_{ij}\, x_j = 1 \quad \forall\, i \in M \tag{4}$$

Lo que implica que un grupo de Clientes puede tener asociado sólo una Acción.

$x_j = \begin{cases} 1 \\ 0 \end{cases}$ $\forall\, j \in N$, x puede tomar los valores 0 ó 1, dependiendo si la Acción se aplica o no al grupo de Clientes.

5.4 Restricciones

5.4.1 Restricción de % de inversión

Sea P_i el porcentaje máximo de inversión sobre el grupo i y GA_i la ganancia para el grupo i

$$\sum_{j=1}^{n} c_j x_j \leq\ P_i GA_i \quad \forall\, i \in M \tag{5}$$

Indica que el costo de cubrir el grupo no puede sobrepasar el porcentaje de inversión definido para el grupo.

Para seleccionar las acciones a aplicar sobre los grupos se utilizará la técnica de Constraint Programming y Metaheurísticas para obtener cual es la mejor técnica que se comporta para tomar esta decisión de optimización de costo, se implementará el modelo en primera instancia en Constraint Programming y luego en Metaheurísticas, particularmente utilizando algoritmos genéticos.

5.5 Constraint Programming

El algoritmo para la utilización Constraint Programming será el siguiente:

```
Inicio
Carga Matriz de posibles coberturas;
Carga vector de costos;
Carga restricción SPP;
        Repetir por cada fila
                Repetir por cada columna
                        Carga posibles coberturas al Solver
                Hasta terminar columnas
        Hasta terminar filas
        Se carga la función objetivo al Solver
        Se lanza el Solver en búsqueda de una solución
        Repetir por cada solución
                Muestra el costo y vector de solución
        Fin Soluciones
Fin
```

Figura N° 5.4.1: Algoritmo Constraint Programming, resolución del problema

Para la implementación del algoritmo de Constraint Programming se está utilizando un Solver provisto por Google llamado Google.OrTools [22].

5.6 Algoritmo Genético

Se utilizará la información que se define en la base del problema, teniendo una matriz que indica el cruce entre acciones/grupos de clientes, cada asociación grupo/cliente tiene un costo asociado, la representación del cromosoma será binaria, donde cada bít indica la cobertura o no de la fila, 0 indica no cobertura, 1 indica cobertura.

Como primera etapa se obtiene la población inicial con cromosomas generados al azar.
Una vez generada la población se evalúan los cromosomas, y se verifica la factibilidad, si algún cromosoma dentro de la población no es una solución factible, esta se repara en base al costo. La reparación es la siguiente:

- Se toma el cromosoma y se verifican que fila faltan por cubrir
- Se genera una lista de variables de decisión que cubren las filas
- Se evalúa el costo de habilitar cada variable de decisión
- Se elige la variable que implique menor costo al habilitar en el cromosoma
- Se comenzará por las variables de decisión que cubre la menor cantidad de filas
- Existe un factor que indica cual es el % máximo de genes que puede tener un cromosoma por repararse

Luego de tener la población inicial con soluciones factibles, se continua con la evaluación de esta población, la selección de los padres para el cruce se realiza mediante la selección por ruleta, donde se generarán dos ruletas, de las cuales se obtendrán los nuevos padres, luego de seleccionar los padres del cruce se evalúa el valor de Pc que corresponde a una probabilidad de cruzamiento para el problema, de cumplirse, estos se cruzarán y darán origen a dos nuevos hijos para generar la próxima generación, de no cumplirse la probabilidad, los padres pasarán directamente a la siguiente generación pero antes de realizarse esta acción se genera una mutación para los nuevos cromosomas.

La mutación de los hijos se realizará bajo una tasa dada como parámetro al algoritmo. El objetivo de mutar los genes permite escapar de valores locales para lograr encontrar un valor global mejor para la solución

La condición de parada del algoritmo dependerá de la cantidad de generaciones definidas como parámetros de entrada al algoritmo.

El pseudocódigo de lo nombrado anteriormente es el siguiente:
Cg=Cantidad de generaciones
Tp=Tamaño población
Fr=Factor de reparación
Pc=Probabilidad de cruce
Pm=Probabilidad de mutación
Elitismo= indica si el proceso se ejecuta con elitismo

```
Inicio
    Mientras Tamaño Población<Tp
        Generar Cromosoma al azar
        Si cromosoma NO Factible y reparación Factible Entonces
            Reparar Cromosoma
        Sino
            Generar Nuevo Cromosoma Factible al Azar
        Fin Si
```

```
        Fin Mientras
        Calcular la función de aptitud de cada individuo de la Población
Mientras Número Generación < Cg
Inicio /* Producir nueva generación */
        Mientras Tamaño Población Nueva<Tp
        Inicio /*Ciclo Reproductivo */
                Si Elitismo Entonces
                        Insertar mejor individuo de generación anterior en nueva población
                Fin Si
                Seleccionar Individuo 1 en base a ruleta
                Seleccionar Individuo 2 en base a ruleta
                Evaluar la probabilidad de cruce de los individuos (Pc)
                Si cumple cruzar Entonces
                        Cruzar individuo 1 con Individuo 2 con cruce de dos puntos
                        obteniendo dos descendientes.
                        Calcular la función de aptitud de cada hijo
                Fin Si
                Evaluar la probabilidad de mutación de los individuos (Pm)
                Si cumple Entonces
                        Mutar Nuevos Individuos
                Fin Si
                Si Solución NO Factible y reparación Factible Entonces
                        REPARAR Cromosoma
                Sino Si NO es Factible Reparar
                        GENERAR NUEVO CROMOSOMA FACTIBLE AL AZAR
                Sino Si es Solución Factible
                        Insertar individuo 1 en nueva población
                        Insertar individuo 2 en nueva población
                Fin Si
        Fin Mientras
Fin Mientras
Mostrar mejor cromosoma
Fin
```

Figura N° 5.4.1: Algoritmo Genético, resolución del problema

Para la implementación del algoritmo genético se está utilizando una librería en C# llamada btl.generic [23].

6 Experimentos

Ambos algoritmos fueron implementados en el lenguaje C# utilizando como IDE de programación Visual Studio 2010, los experimentos están ejecutándose en un equipo con las siguientes características:

Procesador: Intel Core I3
RAM: 4 GB RAM
Sistema Operativo: Windows 7 Home Edition 64 bits
Microsoft .Net Framework 4.0

6.1 Implementación Constraint Programming

Para la implementación del Solver que resuelve el Set Partitioning con la técnica de Constraint Programming se utilizó una librería dispuesta por Google llamada OrTools, la cual es un librería que permite resolver distintos problemas de Constraint Programming con un mínimo esfuerzo de programación.

Para utilizar las propiedades que brinda esta librería se debió cargar la información necesaria para que el algoritmo pudiese trabajar de manera óptima para llegar a un resultado esperado.

En resumen se realizó la lectura y carga de la matriz de posibles soluciones al algoritmo, además de los costos, esto fue lo necesario para que el Solver se ejecutase correctamente y encontrará la mejor solución.

A continuación se grafica lo explicado anteriormente

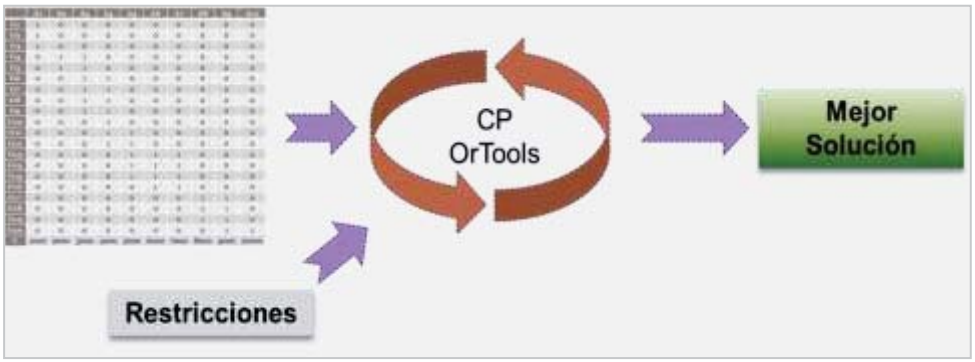


Figura N° 5.4.1: Constraint Programming

6.2 Implementación Algoritmo Genético

Para la implementación del Algoritmo Genético se utilizó una librería llamada btl.generic el cual posee las características básicas para trabajar con algoritmos genéticos (Generación de Cromosomas, Selección, Cruzamiento, Mutación)

A continuación se despliega el flujo del algoritmo genético

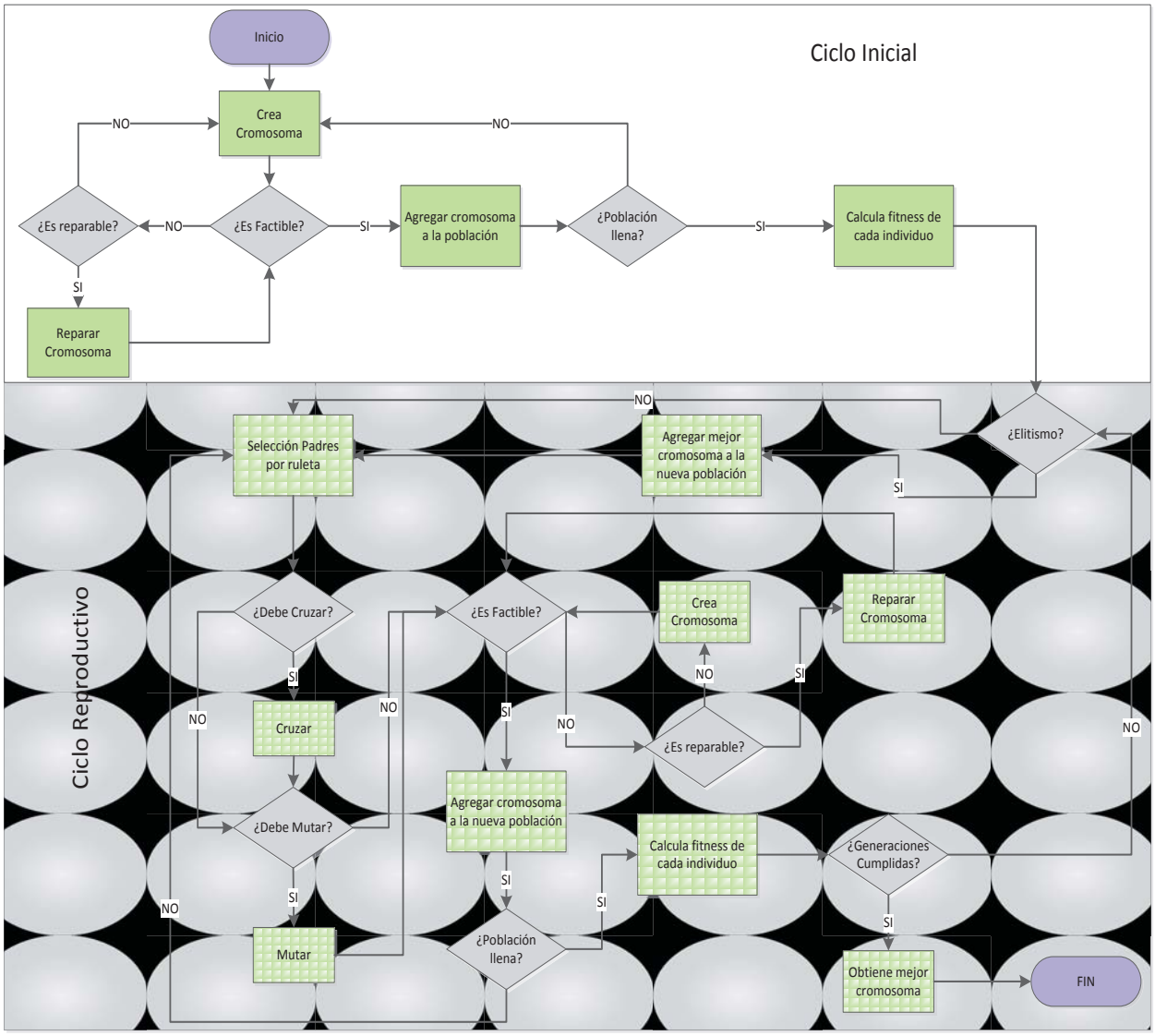


Figura N° 5.4.1: Algoritmo Genético

El cruzamiento realizado corresponde a un cruzamiento de un punto en base a una tasa, si la tasa se cumple, los cromosomas padres se cruzarán, la selección de estos se lleva a cabo mediante la técnica de selección por ruleta

La mutación de igual modo se realiza en base a una tasa, si la tasa se cumple los cromosomas hijos o los padres que no fueron cruzamos se mutan

La verificación de factibilidad de un individuo se realizó de la siguiente manera:

- Se tiene un cromosoma como posible solución factible

1	0	1	0	1	1

- A partir de este cromosoma se genera un vector de coberturas, imaginemos que nuestra matriz es la siguiente:

	1	0	1	0	1	1	
	A1	A2	A3	A4	A5	A6	V
G1	1	0	0	1	0	0	1
G2	1	0	1	0	0	0	2
G3	0	1	0	0	1	0	1
G4	0	1	0	1	1	0	1
G5	0	0	0	0	0	1	1

Figura N° 5.4.12: Reparación, cromosoma no reparable

- A partir de la posible solución obtenida se genera un vector de cobertura, si el vector de cobertura posee una cantidad de sobre coberturas que supera la tasa de reparación, el cromosoma se marca como no factible y se genera uno nuevo al azar.

- Si a partir de la posible solución encontrada se genera un vector de coberturas con valores menores a 1 (sólo 0 y 1), o la cantidad de sobre cobertura en el cromosoma es menor que la tasa de reparación, el cromosoma se marca como reparable

	1	0	0	0	1	1	
	A1	A2	A3	A4	A5	A6	V
G1	1	0	0	0	0	0	1
G2	0	0	1	1	0	0	0
G3	0	1	1	0	1	0	1
G4	0	0	0	0	1	0	1
G5	0	0	0	0	0	1	1

Figura N° 5.4.12.3: Reparación, cromosoma reparable

La reparación de un individuo se realiza de la siguiente manera:

	1	0	0	0	1	1	
	A1	A2	A3	A4	A5	A6	V
G1	1	0	0	0	0	0	1
G2	0	0	1	1	0	0	0
G3	0	1	1	0	1	0	1
G4	0	0	0	0	1	0	1
G5	0	0	0	0	0	1	1

Figura N° 5.4.14: Reparación del cromosoma

Para Cromosomas sin sobre coberturas:

- Se recorre el vector de cobertura, hasta encontrar un grupo no cubierto (posición del vector con valor 0).
- Luego se recorren las columnas de la matriz de posibles soluciones (acciones), y se verifica que columnas cubren el grupo.
- Si existiese más de una columna que cubre al grupo, se seleccionará la que posea menor costo.
- Luego se repite el ciclo de verificación de factibilidad.

Para Cromosomas con sobre coberturas que no superen la tasa de reparación:

- Se recorre el vector de cobertura, hasta encontrar un grupo sobre cubierto (posición del vector con valor > 1).

- Luego se recorren las columnas de la matriz de posibles soluciones (acciones), y se verifica que columnas cubren el grupo que se encuentra sobre cubierto, una vez encontradas, se quitan de la solución.

- Luego se repite el ciclo de verificación de factibilidad.

Una vez explicado cómo funcionan ambos algoritmos se procede a mostrar los resultados de los primeros experimentos, el objetivo de esta primera etapa de experimentación fue seleccionar entre las técnicas de Constraint Programming y Algoritmos Genéticos cual fue la que se comportó de mejor manera para resolver el problema en cuestión.

6.3 Primer Ciclo de Experimentos

Para la verificación de que ambos algoritmos resolvieran de manera óptima el problema se utilizó en primera instancia una matriz pequeña de 20 Grupos x 10 Acciones, como la que se aprecia a continuación:

	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10
G1	1	0	0	0	0	0	0	0	0	0
G2	1	0	0	0	0	0	0	0	0	0
G3	1	0	0	0	0	0	0	0	0	0
G4	0	1	1	0	0	0	0	0	0	0
G5	0	1	1	0	0	0	0	0	0	0
G6	0	0	1	1	0	0	0	0	0	0
G7	0	0	1	1	0	0	0	0	0	0
G8	0	0	1	1	0	0	0	0	0	0
G9	0	0	1	1	0	0	0	0	0	0
G10	0	0	0	1	0	0	0	0	0	0
G11	0	0	0	1	1	0	0	0	0	0
G12	0	0	0	1	1	0	0	0	0	0
G13	0	0	0	0	1	1	1	0	0	0
G14	0	0	0	0	1	1	1	0	0	0
G15	0	0	0	0	1	1	1	0	0	0
G16	0	0	0	0	0	1	1	0	0	0
G17	0	0	0	0	0	0	0	1	1	0

G18	0	0	0	0	0	0	0	1	1	0
G19	0	0	0	0	0	0	0	1	1	0
G20	0	0	0	0	0	0	0	0	1	1
C	1000	2000	3000	4000	5000	6000	7000	8000	9000	10000

Figura N° 5.4.1: Matriz 20x10 Experimentos

Las columnas indican las acciones (A1 al A10) y en las filas los grupos a cubrir (G1 al G20), y bajo la fila de los grupos existe una fila que se denomina C la cual es un vector de costos correspondiente a cada acción.

Se muestran en color gris las posibles soluciones.

Solución 1: A1-A2-A4-A6-A8-A10 (1101010101) → Costo: 31.000.-

	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10
G1	1	0	0	0	0	0	0	0	0	0
G2	1	0	0	0	0	0	0	0	0	0
G3	1	0	0	0	0	0	0	0	0	0
G4	0	1	1	0	0	0	0	0	0	0
G5	0	1	1	0	0	0	0	0	0	0
G6	0	0	1	1	0	0	0	0	0	0
G7	0	0	1	1	0	0	0	0	0	0
G8	0	0	1	1	0	0	0	0	0	0
G9	0	0	1	1	0	0	0	0	0	0
G10	0	0	0	1	0	0	0	0	0	0
G11	0	0	0	1	1	0	0	0	0	0
G12	0	0	0	1	1	0	0	0	0	0
G13	0	0	0	0	1	1	1	0	0	0
G14	0	0	0	0	1	1	1	0	0	0
G15	0	0	0	0	1	1	1	0	0	0
G16	0	0	0	0	0	1	1	0	0	0
G17	0	0	0	0	0	0	0	1	1	0
G18	0	0	0	0	0	0	0	1	1	0
G19	0	0	0	0	0	0	0	1	1	0
G20	0	0	0	0	0	0	0	0	1	1
C	1000	2000	3000	4000	5000	6000	7000	8000	9000	10000

Figura N° 5.4.12: Matriz 20x10 Experimentos primera solución

Solución 2: A1-A2-A4-A6-A9 (1101010010) → Costo: 22.000.-

	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10
G1	1	0	0	0	0	0	0	0	0	0
G2	1	0	0	0	0	0	0	0	0	0
G3	1	0	0	0	0	0	0	0	0	0
G4	0	1	1	0	0	0	0	0	0	0
G5	0	1	1	0	0	0	0	0	0	0
G6	0	0	1	1	0	0	0	0	0	0
G7	0	0	1	1	0	0	0	0	0	0
G8	0	0	1	1	0	0	0	0	0	0
G9	0	0	1	1	0	0	0	0	0	0
G10	0	0	0	1	0	0	0	0	0	0
G11	0	0	0	1	1	0	0	0	0	0
G12	0	0	0	1	1	0	0	0	0	0
G13	0	0	0	0	1	1	1	0	0	0
G14	0	0	0	0	1	1	1	0	0	0
G15	0	0	0	0	1	1	1	0	0	0
G16	0	0	0	0	0	1	1	0	0	0
G17	0	0	0	0	0	0	0	1	1	0
G18	0	0	0	0	0	0	0	1	1	0
G19	0	0	0	0	0	0	0	1	1	0
G20	0	0	0	0	0	0	0	0	1	1
C	1000	2000	3000	4000	5000	6000	7000	8000	9000	10000

Figura N° 5.4.13: Matriz 20x10 Experimentos segunda solución

Como se logra observar el problema de la matriz 20X10 tiene dos soluciones posibles, la primera con un costo de 31.000 y la segunda con un costo de 22.000-

Para este caso tan pequeño ambos algoritmos llegaron siempre a la solución con mejor costo.

Luego de verificar que ambos algoritmos funcionaran correctamente se realizaron pruebas con matrices de los siguientes tamaños:

- 40 Grupos X 20 Acciones
- 80 Grupos X 20 Acciones
- 160 Grupos X 40 Acciones

Para cada matriz incluyendo la primera, se realizaron dos iteraciones de experimentos, la primera consistió en ejecutar manualmente el algoritmo y la segunda se realizó mediante un ciclo, en la primera iteración se ejecutaron diez experimentos con distintas configuraciones y en la segunda treinta experimentos, a continuación se comparan gráficamente la resolución del algoritmo genético en base a la modificación de los parámetros que rigen el algoritmo. Cada serie posee distinta configuración de parámetros como se muestra en la siguiente imagen:

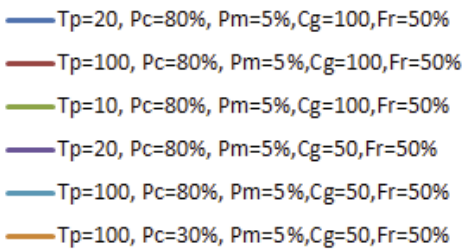


Figura N° 5.4.14: Experimentos, configuración Algoritmo Genético

La nomenclatura es la siguiente:

Tp: Tamaño de la población
Pc: Probabilidad de cruzamiento
Pm: Probabilidad de mutación
Cg: Cantidad de generaciones
Fr: Factor de reparación

6.3.1 Comparación Configuraciones Algoritmo Genético basado en el tiempo de resolución

La primera evaluación realizada se hizo en base al tiempo de resolución del algoritmo en base a distintas configuraciones, como se aprecia en el gráfico para tamaños de población pequeños el algoritmo resuelve rápidamente, mostrando un valor más elevado al comienzo del algoritmo el cual incluye la carga de la información en memoria

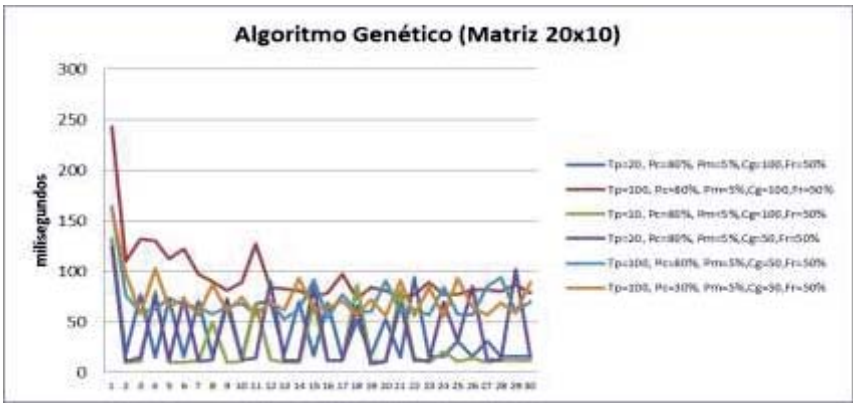


Figura N° 6.3.1.1: Experimentos, Genético (Matriz 20X10) ms

Se aprecia que las resoluciones con poblaciones pequeñas aunque un poco mayores se mantienen en un tiempo de procesamiento bajo, pero ya comienza a marcarse una diferencia de comportamiento, estos se dividen en dos grupos, los más altos son los que poseen los mayores tamaños de población, por el contrario los de resolución en tiempos menores poseen poblaciones pequeñas.

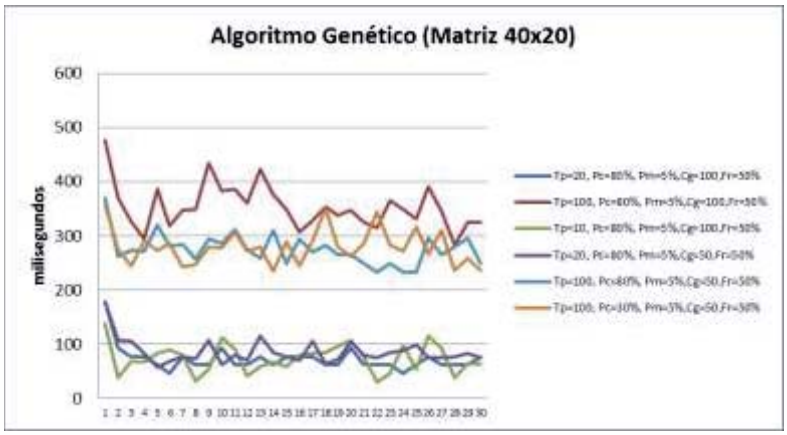


Figura N° 6.3.1.2: Experimentos, Genético (Matriz 40X20) ms

A medida que la cantidad de datos a ser procesado crece, el comportamiento de los algoritmos se mantiene incrementando su tiempo de resolución, como se aprecia en la gráfica, se siguen marcando las dos tendencias y aumentando el tiempo de resolución el algoritmo que posee una configuración con mayor cantidad de generaciones

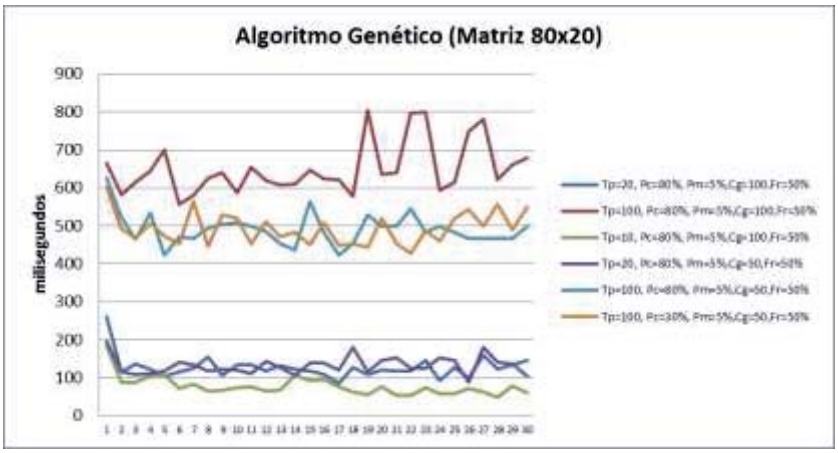


Figura N° 6.3.1.3: Experimentos, Genético (Matriz 80X20) ms

Para la matriz más grande con la cual se experimentó, se aprecia la misma tendencia, para algoritmos con poblaciones pequeñas la resolución es la más rápida pero se sacrifican los buenos resultados como se verá más adelante, a diferencia de los algoritmos con poblaciones más numerosas

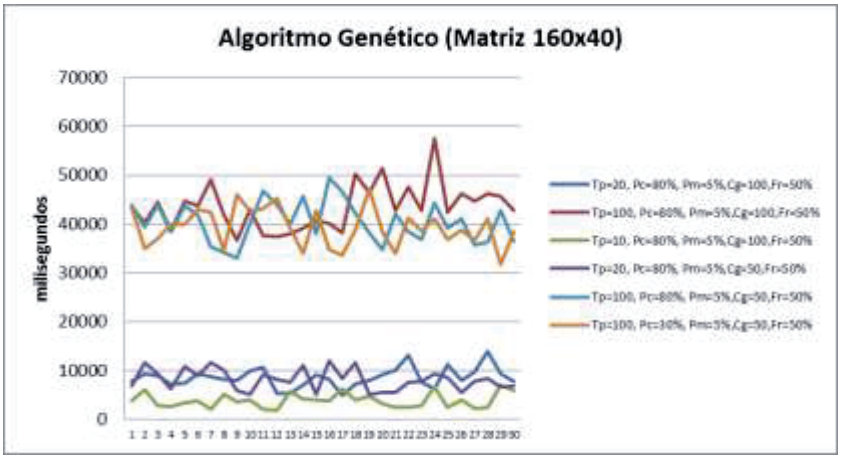


Figura N° 6.3.1.4: Experimentos, Genético (Matriz 160X40) ms

6.3.2 Comparación Algoritmo Genético en base a memoria utilizada

En la primera tanda de experimentos se apreció una gran variabilidad de utilización de memoria al ejecutar el algoritmo, pero esto se debió a que cada vez que se ejecutaba el algoritmo se cargaba la matriz en memoria, por ende la lectura/escritura disparó los valores. Por lo mismo no se tomó en cuenta esta medición específicamente para la primera tanda de experimentos tomando en cuenta los resultados obtenidos en la segunda iteración.

A continuación se muestra la variabilidad entre la utilización de memoria que arrojó cada algoritmo.

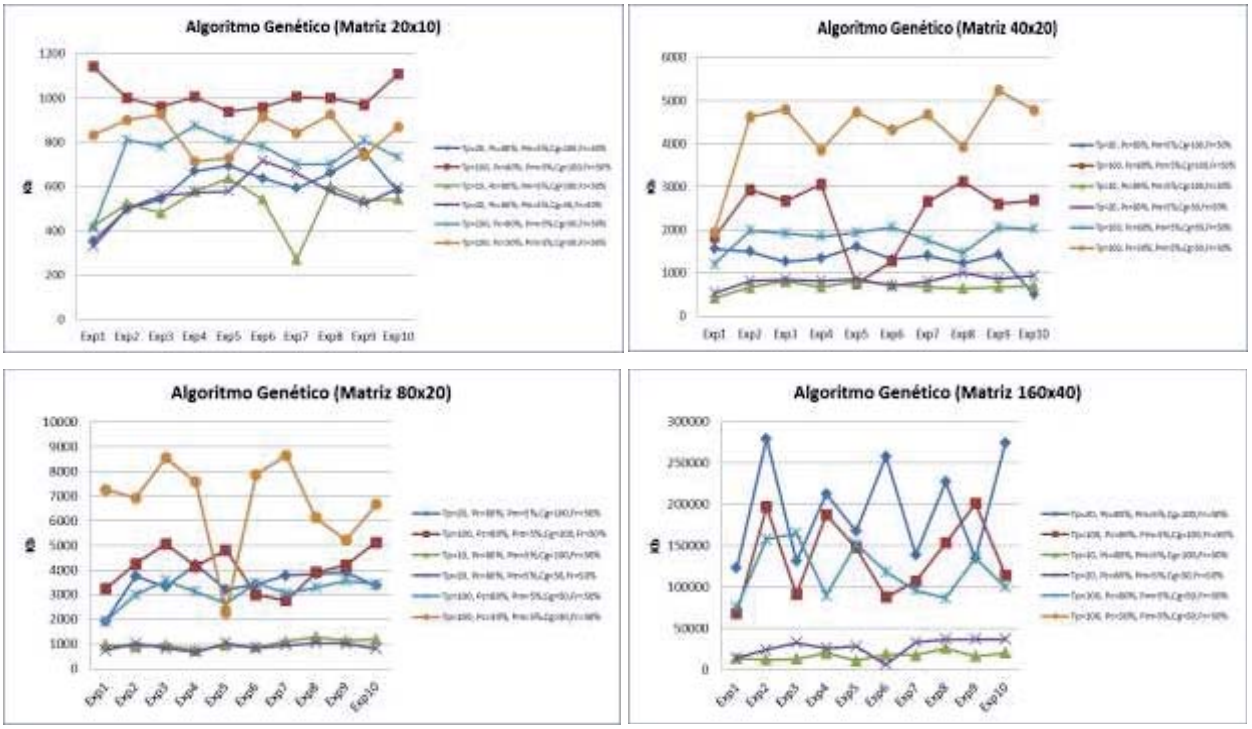
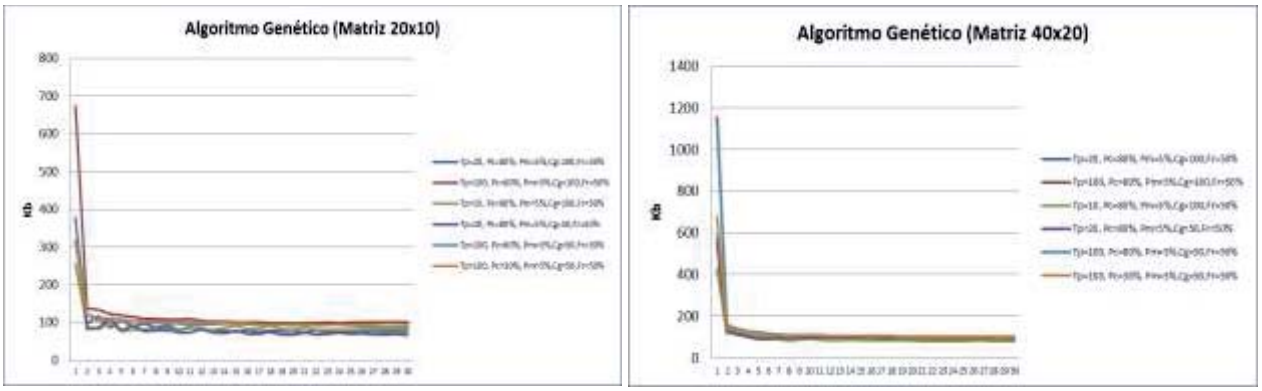


Figura N° 6.3.2.1: Experimentos, Genéticos uso de memoria primera iteración

Para la segunda iteración, la comparación del algoritmo con distintos parámetros no develó una diferencia notoria de comportamiento, en todos los casos la mayor carga se lleva al comienzo del proceso en el cual se carga la matriz de posibles coberturas y los costos al algoritmo.

Como se puede apreciar en los siguientes gráficos.



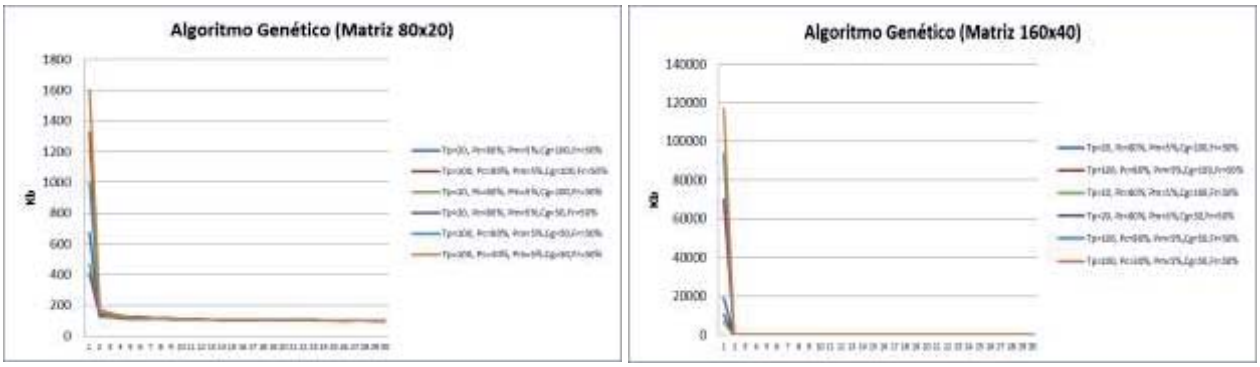


Figura N° 6.3.2.2: Experimentos, Genéticos uso de memoria segunda iteración

6.3.3 Comparación Algoritmo Genético en base al Costo obtenido

Se puede apreciar que el algoritmo configurado con menor cantidad de población es el que entrega la mayor cantidad de resultados deficientes, por el contrario los algoritmos con poblaciones mayores entregaron el mejor costo en todos los casos, es importante recalcar que la cantidad de datos a procesar es pequeñisima

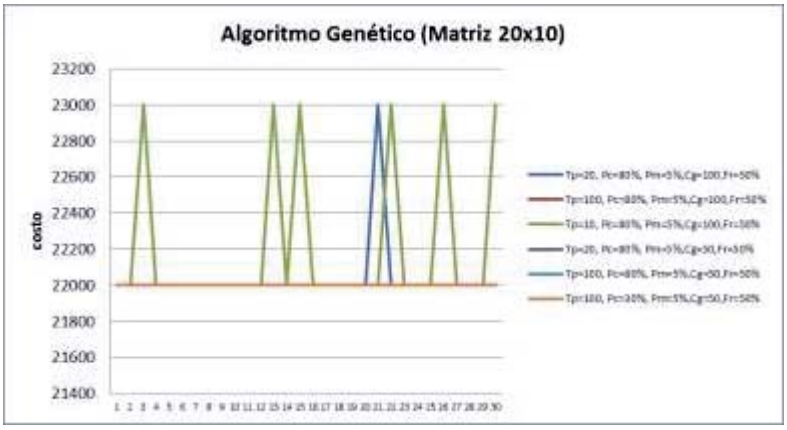


Figura N° 6.3.3.1: Experimentos, Costo Genético (Matriz 20X10)

Al igual que en el gráfico anterior la tendencia se mantiene, se aprecia que los algoritmos con poblaciones pequeñas entregan soluciones de menor calidad, aunque existe una excepción en un algoritmo cuya cantidade población esta en las mayores, esto está relacionado a la tasa de cruzamiento y cantidad de generaciones que se configuró en ese momento, ya que las otras dos configuraciones se diferencian en, para la primera una cantidad mayor de generaciones y para la segunda una tasa menor de cruzamiento, lo que lleva a que el algoritmo explore menos y genere una mayor cantidad de cromosomas homogeneos

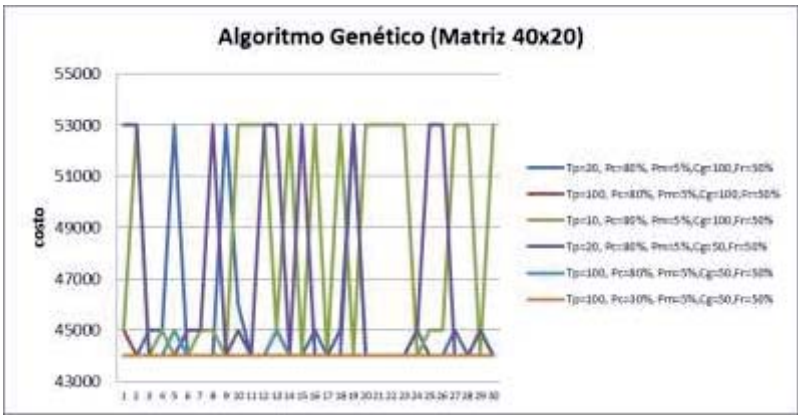


Figura N° 6.3.3.2: Experimentos, Costo Genético (Matriz 40X20)

Como se puede apreciar la tendencia del gráfico anteriór sigue su curso, y se repite en el siguiente gráfico, ocurre lo mismo en relacion a la cantidad de población, generaciones y tasa de cruzamiento

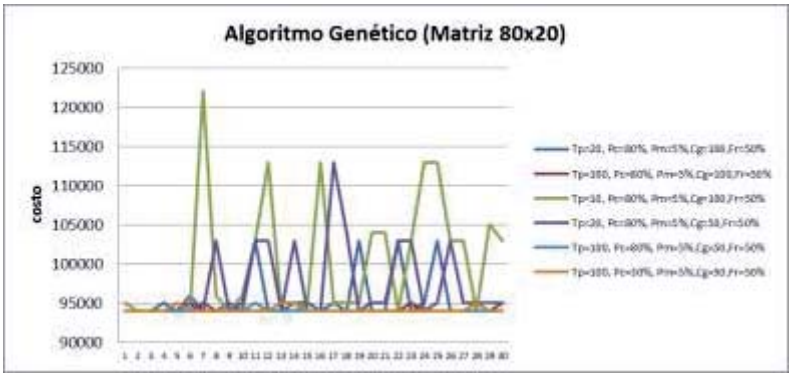


Figura N° 6.3.3.3: Experimentos, Costo Genético (Matriz 80X20) Kb

En el siguiente gráfico se acentúa aún más las fluctuaciones ya que existe una mayor cantidad de soluciones con diferentes costos, pero se aprecia que los algoritmos con poblaciones numerosas se apoderan de las mejores soluciones entregadas por el algoritmo genético.

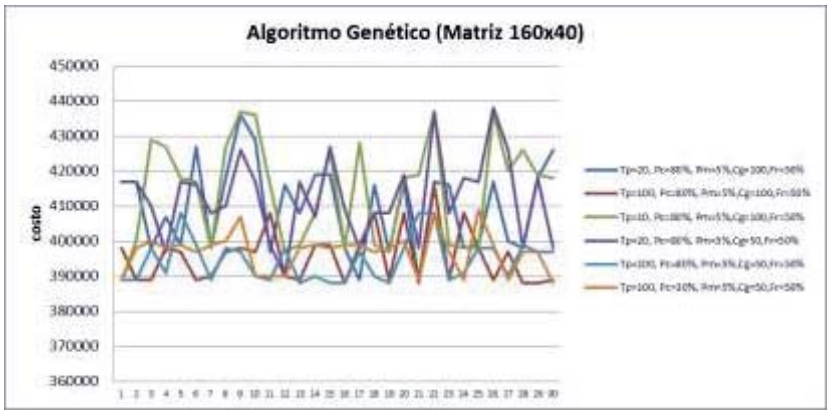


Figura N° 6.3.3.4: Experimentos, Costo Genético (Matriz 160X40) Kb

Teniendo los tiempos de resolucion, uso de memoria y la calidad de las soluciones entregadas para el problema resuleto por algoritmos genéticos, existen tres configuraciones que fueron candidatas a se seleccionadas, las cuales son:

Tp=100, Pc=80%, Pm=5%,Cg=100,Fr=50%
Tp=100, Pc=80%, Pm=5%,Cg=50,Fr=50%
Tp=100, Pc=30%, Pm=5%,Cg=50,Fr=50%

Figura N° 6.3.3.5: Experimentos, Configuración Algoritmo Genético

Los promedios de tiempo, memoria y costos de las soluciones para la matriz de mayor tamaño son:

Tp=100, Pc=80%, Pm=5%,Cg=100,Fr=50%			Tp=100, Pc=80%, Pm=5%,Cg=50,Fr=50%			Tp=100, Pc=30%, Pm=5%,Cg=50,Fr=50%		
ms	kb	costo	ms	Kb	costo	ms	kb	costo
43440	2441	395633	40379	3232	394567	39267	4009	396900

Tabla N° 6.3.3.1: Promedio tiempos, memoria y costos Algoritmo Genético

Lo que se busca encontrar son las mejores soluciones, por ende el algoritmo elegido es el que posee la configuración:

Tp=100, Pc=80%, Pm=5%,Cg=50,Fr=50%

Ya seleccionado los parámetros del Algoritmo Genético, se procede a comparar en los mismos términos con la técnica de Contratint Programming.

6.3.4 Comparación Algoritmo Genético vs Constraint Programming

Para realizar esta comparación se ejecutaron cincuenta experimentos para cada técnica, a modos de compararlos gráficamente.

La primera comparación se basa en el tiempo en milisegundos que se demoró cada técnica en resolver el problema

Como se aprecia en el gráfico con pocos datos ya comienza a apreciarse una diferencia de tiempo de resolución

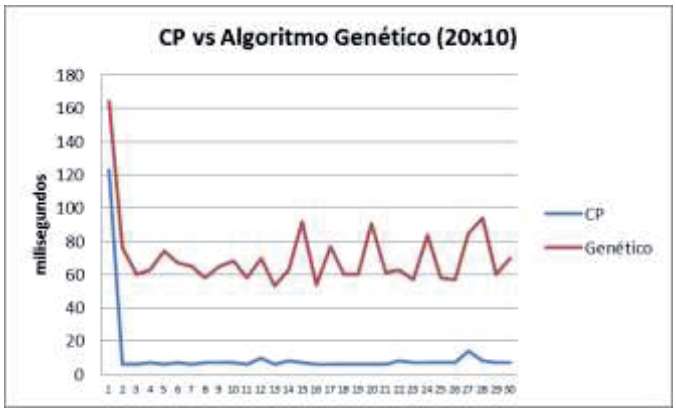


Figura N° 6.3.4.1: Experimentos, CP vs Genético (Matriz 20X10) ms

A medida que se avanzó en las pruebas esta diferencia de tiempo fue haciendo más notoria, como se puede apreciar la resolución con Constraint Programming se mantuvo certa de los 15 ms mientras que la resolución con Algoritmo Genético se elevo excesivamente

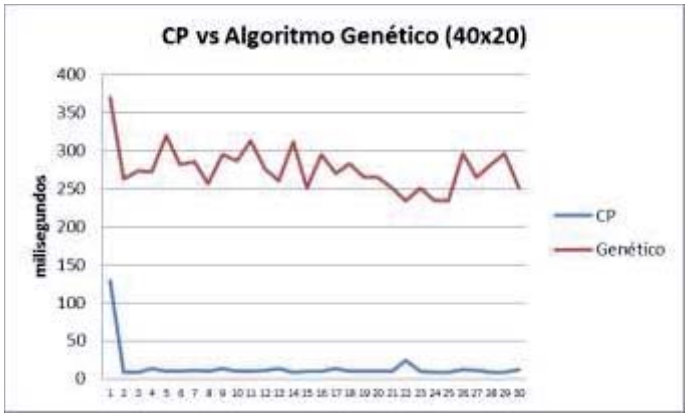


Figura N° 6.3.4.2: Experimentos, CP vs Genético (Matriz 40X20) ms

A continuación se aprecia como se acetúa aún más la diferencia, mientras la resolución con Contrating Programming se mantuvo en un tiempo bajo la busqueda de soluciones con el Algoritmo Genético se elevó a los 500 ms aproximadamente

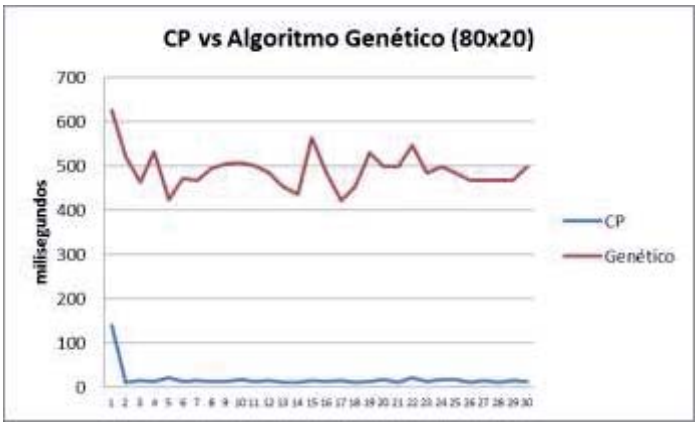


Figura N° 6.3.4.3: Experimentos, CP vs Genético (Matriz 80X20) ms

Como se aprecia para resolver el problema con mayor cantidad de datos los tiempos para el algoritmo genético se dispararon, mientras que para Contraint Programming se mantuvieron bajos, incluso para generar este gráfico se tuvo que cambiar la tasa del eje Y ya que el tiempo de resolución para el Algoritmo Genético subieron a 40 segundos aproximadamente

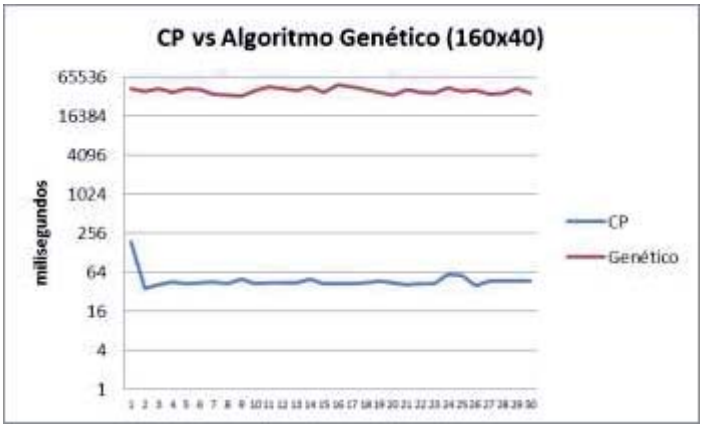


Figura N° 6.3.4.4: Experimentos, CP vs Genético (Matriz 160X40) ms

Luego de comparar los tiempos de resolución a continuación se compara la cantidad de memoria utilizada por cada algoritmo para resolver el problema, al igual que en la segunda iteración de pruebas para el Algoritmo Genético la cantidad de memoria utilizada está directamente relacionada a la lectura de la información inicial, ya que en la

primera iteración de experimentos la lectura de la información se realiza en cada experimento, a diferencia de la segunda la cual carga en primera instancia la información y luego ejecuta los algoritmo de resolución.

A continuación muestra la siguiente gráfica en la cual se aprecia la diferencia explicada.

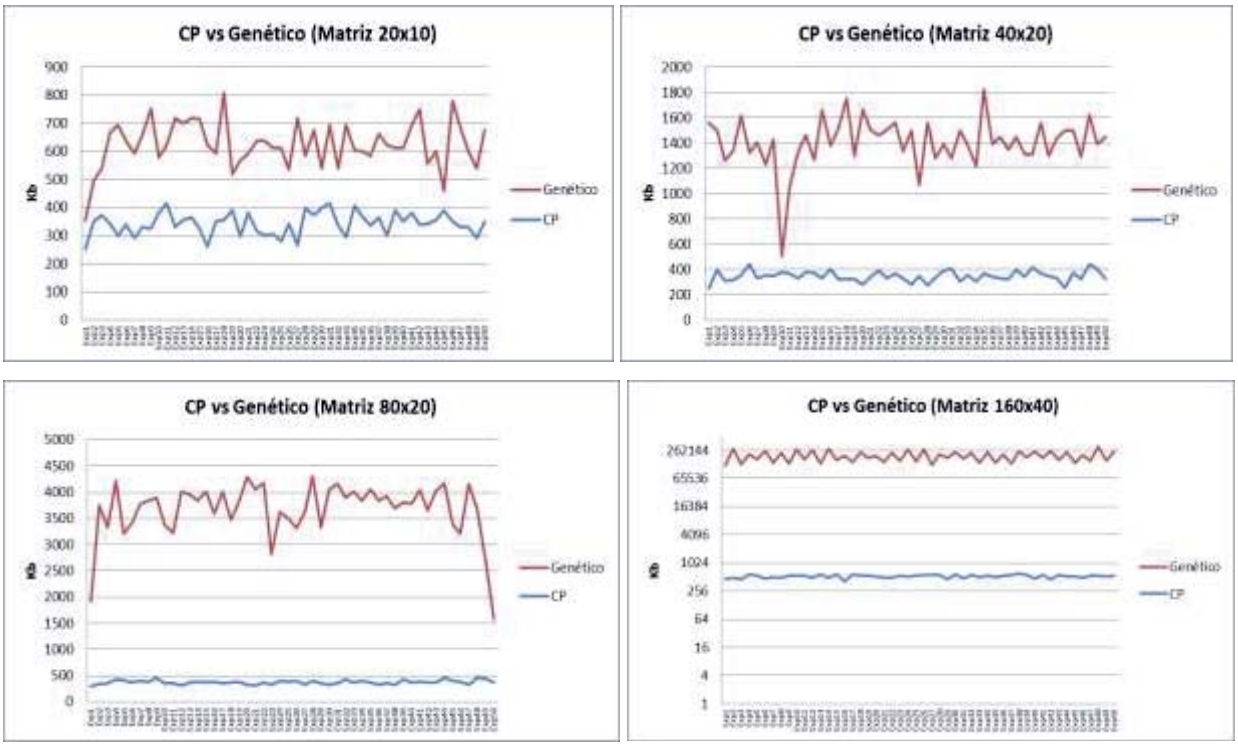
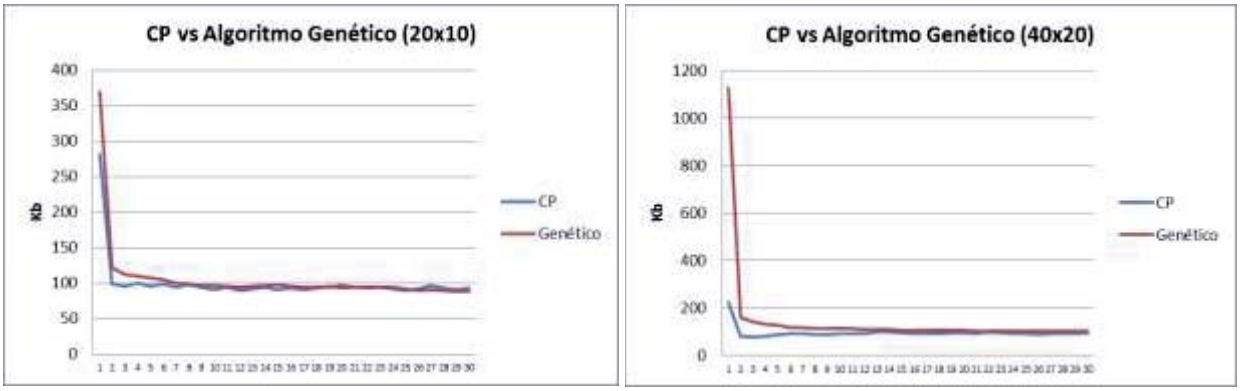


Figura N° 6.3.4.5: Experimentos, CP vs Genéticos uso de memoria primera iteración



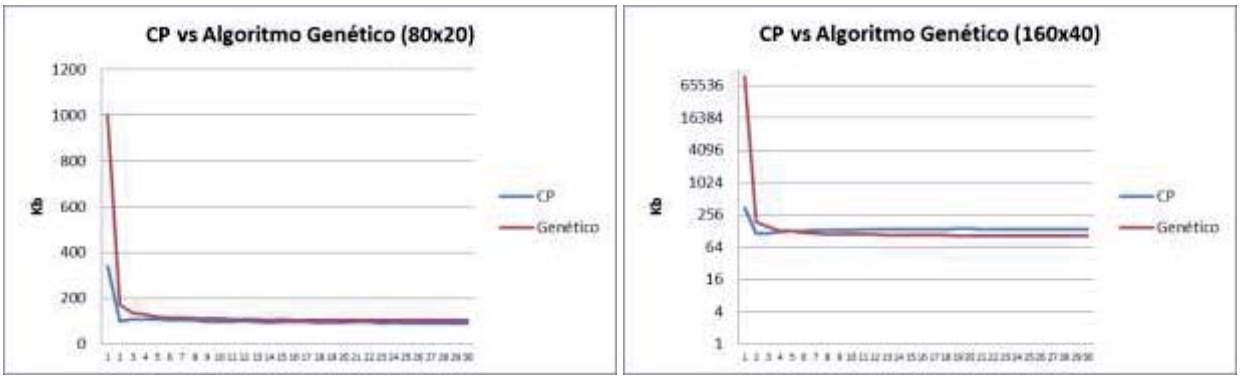


Figura N° 6.3.4.6: Experimentos, CP vs Genéticos uso de memoria segunda iteración

Luego de analizar la cantidad de memoria utilizada por cada algoritmo, se analiza del punto de vista de la calidad de las soluciones entregadas por cada algoritmo

Para la primera instancia de pruebas (la más pequeña), ambos algoritmos entregaron la mejor solución

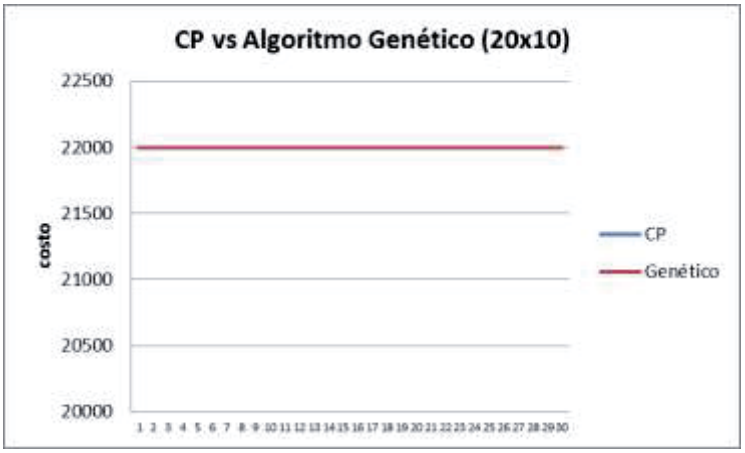


Figura N° 6.3.4.7: Experimentos, Costo CP vs Costo Genético (Matriz 20X10)

Con la segunda fuente de datos (un poco más grande), el Algoritmo Genético entregó dos soluciones con menor calidad que Constraint Programming, como se aprecia en la gráfica

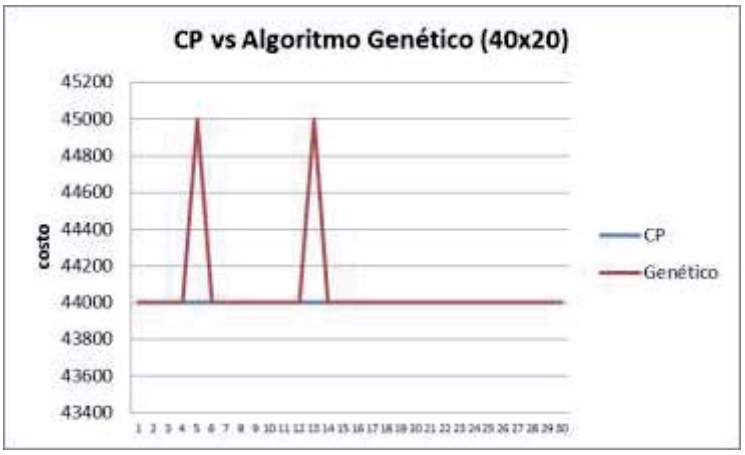


Figura N° 6.3.4.8: Experimentos, Costo CP vs Costo Genético (Matriz 40X20)

Algo muy similar ocurre al realizar los experimentos con la tercera fuente de datos, un poco más grande que la anterior, en esta ocasión el Algoritmo Genético entregó una solución de menor calidad

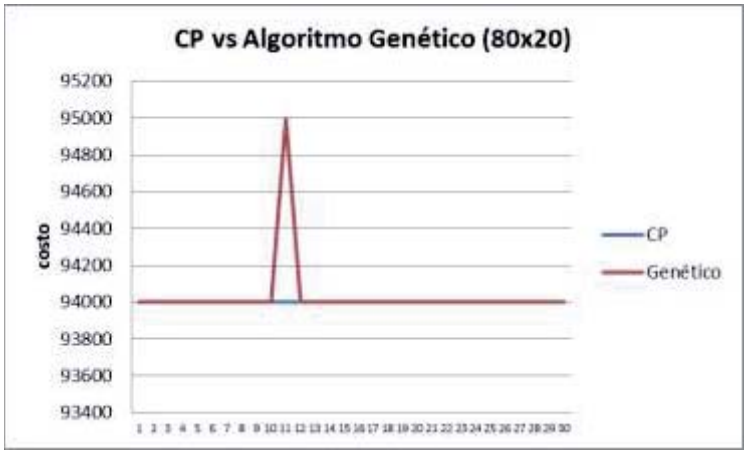


Figura N° 6.3.4.9: Experimentos, Costo CP vs Genético (Matriz 80X20)

Veamos lo que ocurre cuando se tiene una mayor cantidad de datos a procesar, se aprecia claramente que el Algoritmo Genético no compite con la técnica de Constraint Programming que nos entrega siempre la mejor solución dado que es una técnica completa.

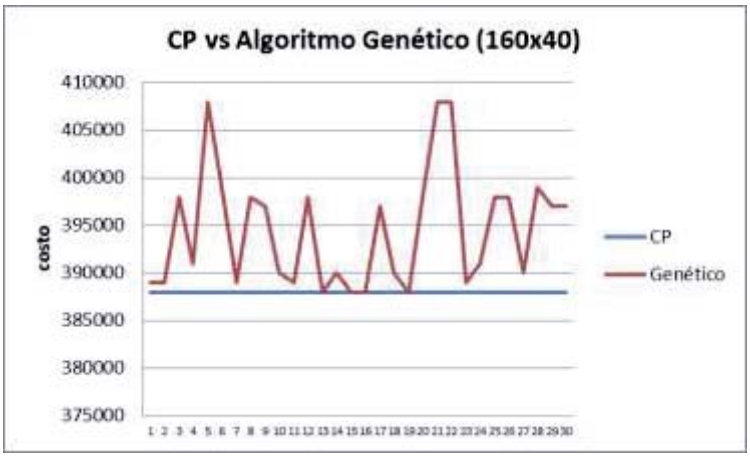


Figura N° 6.3.4.10: Experimentos, Costo CP vs Costo Genético (Matriz 160X40)

6.4 Segundo ciclo de Experimentos

Luego de realizar la primera fase de experimentos, obteniendo como resultado una gran diferencia de resolución entre una técnica y otra, se procedió a una segunda fase de experimentación, en la cual ambas técnicas se utilizar para tratar de resolver el Set Partitioning en base a los siguientes Benchamarks de Beasley [24].

- Sppaa01, provee de 823 filas por 8904 columnas
- Sppkl01, provee de 55 filas por 7479 columnas
- Sppnw01, provee de 135 filas por 51975 columnas
- Sppnw02, provee de 145 por 87879 columnas

En todos los casos ambos algoritmos que implementan la técnica correspondiente no lograron resultados exitosos luego buscar resultados por aproximadamente 30 horas.

Es importante destacar que los Benchmark de Beasley no se asemejan al set de datos del problema particular a resolver, ya que este cuenta con una cantidad de datos reducida y sólo se quería verificar el comportamiento de ambos algoritmos con grandes volúmenes de datos, lo que no impide que ambos algoritmos puedan testearse con datos del problema real en la siguiente fase.

6.5 Tercer ciclo de Experimentos

Una vez realizada la segunda fase de experimentación, se procedió a aplicar ambos algoritmos a datos reales, de dos distintas maneras.

La matriz original de datos posee dos columnas al inicio, la cual indican el gasto realizado por el cliente en promedio (GA) y un porcentaje de inversión (PI) que corresponde al monto máximo a invertir en las acciones aplicables a los clientes, además de los grupos de clientes (G) y las acciones aplicables (A)

GA	PI%	G	A1	A2	A3	A4	A5
20000	10	G1	1	1	0	0	0
30000	20	G2	1	1	0	0	0
40000	25	G3	1	1	0	0	0
50000	30	G4	1	1	0	0	0
60000	35	G5	1	0	0	1	0
70000	40	G6	0	0	1	1	1
80000	45	G7	0	0	1	1	1
90000	50	G8	0	0	0	1	1
100000	55	G9	0	0	0	1	1
150000	60	G10	0	0	0	1	1
		C	2000	3000	4000	5000	6000

Figura N° 6.3.41: Matriz original

Por ende la acción A2 no podría aplicarse al G1, ya que la inversión que se puede aplicar a este Grupo tiene un tope de 2000 (20000 * 10%)

Explicado lo anterior, las dos maneras de resolución fueron la siguiente:

Para la técnica de Algoritmos Genéticos:

- Se agrega la restricción en la fase de verificación de factibilidad de la solución, con esto se verifica que no se puedan cubrir grupos de clientes que no superen el monto máximo de inversión.

- Se pre procesa la matriz, generando una nueva matriz de posibles coberturas en memoria con la restricción ya aplicada, igual al proceso ejecutado para Constraint Programming, luego se ejecuta el algoritmo

Para la técnica de Constraint Programming:

- Se agrega el costo máximo de inversión como restricción adicional al del Set Partitioning al Solver y se ejecuta para que busque las soluciones

- Se pre procesa la matriz, generando una nueva matriz en memoria con la restricción ya aplicada, es decir se quitan todas las coberturas que no son factibles dado el monto máximo de inversión y luego se ejecuta el Solver

6.5.1 Resultados

Algoritmo Genético

Los experimentos realizados con el Algoritmo Genético para ambas maneras de resolución resultaron fallidos, dado que en ambos casos el algoritmo se ejecutó por más de 30 horas y no encontró resultados.

Revisando en mayor profundidad, el costo mayor del algoritmo se lo llevó la reparación de cromosomas, lo cual no permitió llegar a resultados en el tiempo esperado.

Constraint Programming

Los experimentos realizados con esta técnica resultaron exitosos, y el Solver obtuvo resultados en poco tiempo.

A continuación se muestran los resultados de la resolución con esta técnica.

Para una matriz de 40 acciones por 80 grupos de clientes, el pre procesamiento de la matriz logra bajar los tiempos de resolución y la cantidad de memoria utilizada para encontrar las soluciones, como se puede apreciar en la gráfica siguiente.

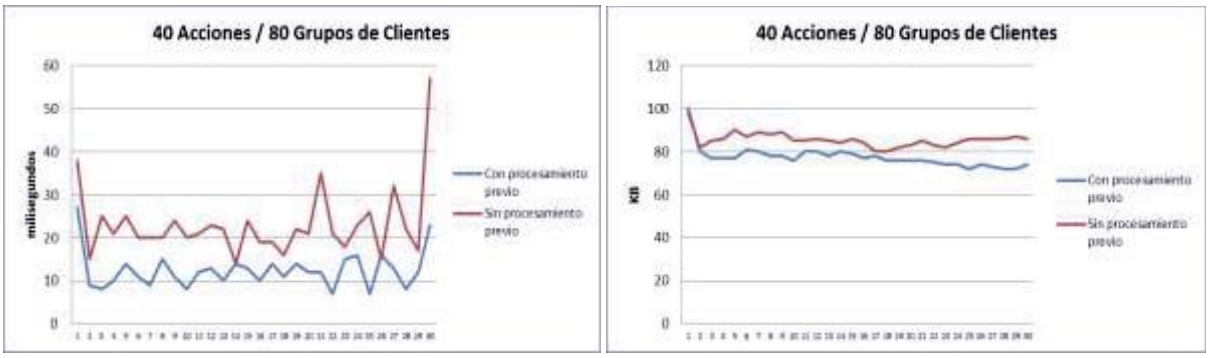


Figura N° 6.5.1.1: Comparación resolución con y sin pre procesamiento (matriz 80X40)

Luego para una matriz un poco mayor en cuanto a grupos de clientes, la diferencia de tiempo y memoria se hace más notoria, como se aprecia a continuación.

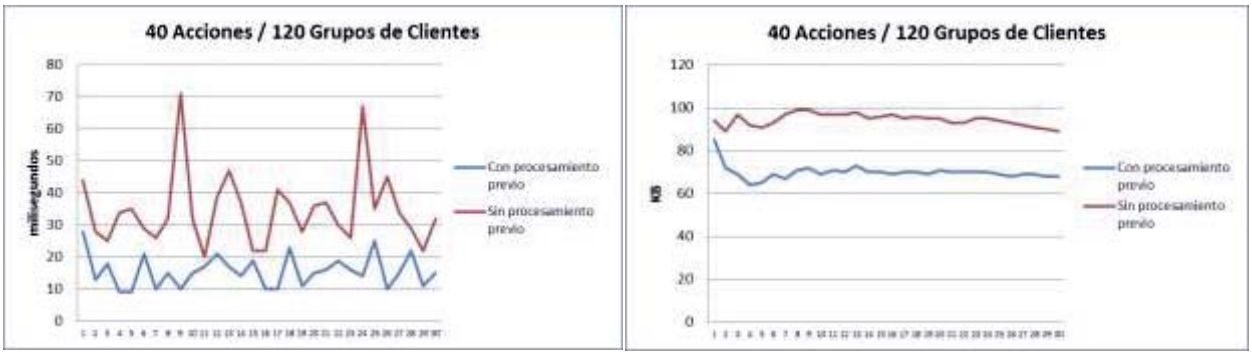
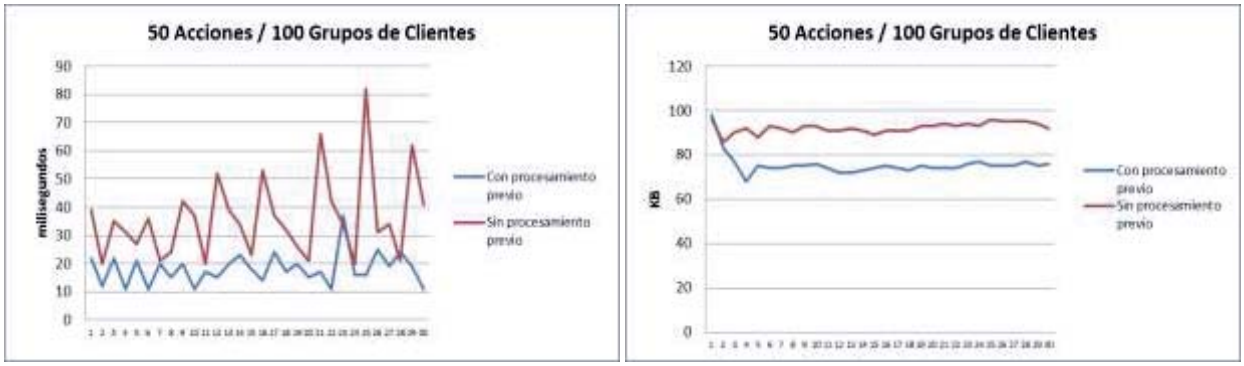


Figura N° 6.5.1.2: Comparación resolución con y sin pre procesamiento (matriz 120X40)

El comportamiento se mantiene a medida que la matriz crece, como se aprecia en la gráfica siguiente, el pre procesamiento de la matriz continua siendo más óptimo que agregar las restricciones al Solver de Constraint Programming y que este resuelva.



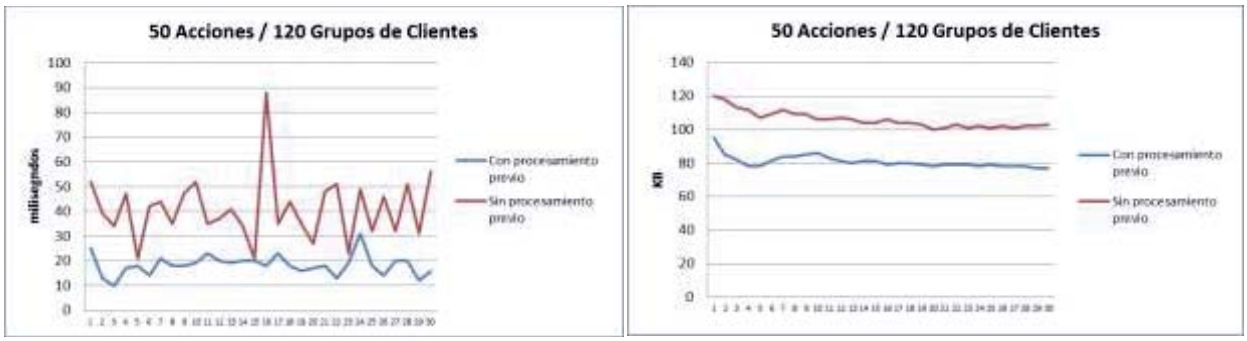


Figura N° 6.5.1.3: Comparación resolución con y sin pre procesamiento (matriz 100X50 y 120X50)

Luego de realizar la serie de experimentos ya mencionados y desplegados los resultados, se tienen las bases claras para seleccionar la técnica a utilizar desde acá en adelante para resolver el problema.

Como se pudo observar en los experimentos, los tiempos de resolución, utilización de memoria y calidad de las soluciones entregadas siempre fueron las mejores en el caso de la utilización de la técnica de Constraint Programming, por lo mismo, se decidió utilizar esta técnica para resolver el problema planteado.

7 Aplicación actual vs Nuevo modelo planteado

En la actualidad, el cruce de información mediante las planillas Excel, toma aproximadamente dos horas de trabajo, requiere este tiempo debido a que es una labor que debe hacerse de manera minuciosa, para minimizar la probabilidad de errores.

La entrega de acciones a grupos de clientes, se sobrepone en un 30% aproximadamente, llevando esto a números, ejemplo, en una campaña pasada se realizaron acciones a grupos de clientes por un total de $90.000.000.-, siendo que existían 30 grupos de clientes, por ende se sobre cubrió a 9 grupos, lo que equivale a un sobre gasto realizado por $20.000.000.- aproximadamente.

Con el nuevo método planteado en base al modelo del Set Partitiong, si se aplicase a la misma campaña, se gastaría aproximadamente 1 hora de trabajo en generar la matriz de posibles soluciones, agregándole el % máximo de inversión y la ganancia, luego el procesamiento de esta por el Solver de Constraint Programming, no demora más de un minuto (ya que son matrices relativamente pequeñas), el resultado arroja una asignación sin sobre coberturas, dada la naturaleza de cómo se planteó el problema (Set Partitioning) , lo que ahorrarían $20.000.000.- a la campaña, el costo del tiempo incurrido para la generación de los datos de entrada es marginal con respecto al gasto de la campaña.

A continuación se muestra una campaña real, realizada en el mes de noviembre del año 2012, llamada “imperdibles”, lo que lleva como base el regalo de cupones de juegos a distintos grupos de clientes, la cantidad de grupos seleccionados fue de 20

Las acciones que se definieron fueron las siguientes:

N° Acción	Descripción de la acción
1	Cupón de juego de $2.000.-
2	Cupón de juego de $3.000.-
3	Cupón de juego de $5.000.-
4	Cupón de juego de $7.000.-
5	Cupón de juego de $10.000.-
6	Cupón de juego de $12.000.-
7	Cupón de juego de $13.000.-
8	Cupón de juego de $15.000.-
9	Cupón de juego de $17.000.-
10	Cupón de juego de $20.000.-
11	Cupón de juego de $22.000.-
12	Cupón de juego de $23.000.-
13	Cupón de juego de $25.000.-
14	Cupón de juego de $27.000.-
15	Cupón de juego de $30.000.-
16	Cupón de juego de $32.000.-
17	Cupón de juego de $35.000.-
18	Cupón de juego de $40.000.-
19	Cupón de juego de $45.000.-
20	Cupón de juego de $50.000.-

Tabla N° 7.1: Acciones

En la siguiente tabla, se compara la asignación de acciones a grupos como se hace de la manera actual versus la asignación de acciones mediante el modelo planteado.

N° Grupo	Cantidad Clientes en Grupo	Asociación de Acción de forma actual	Costo total de la acción	Asociación mediante metodología propuesta	Costo total de la acción
1	1.200	A1	2.400.000.-	A1	2.400.000.-
2	1.150	A1- A2	2.300.000 – 3.450.000.-	A1	2.300.000.-
3	1.100	A3	5.500.000.-	A2	3.300.000.-
4	1.050	A4	7.350.000.-	A3	5.250.000.-
5	980	A5-A6	9.800.000.- – 11.760.000.-	A4	6.860.000.-
6	910	A7	11.830.000.-	A6	10.920.000.-

7	880	A8	13.200.000.-	A8	13.200.000.-
8	840	A9	14.280.000.-	A8	12.600.000.-
9	810	A10-A11	16.200.000.- – 17.820.000.-	A9	13.770.000.-
10	740	A12	17.020.000.-	A9	12.580.000.-
11	700	A13	17.500.000.-	A10	14.000.000.-
12	650	A14	17.550.000.-	A13	16.250.000.-
13	630	A15-A16	18.900.000.- – 20.160.000.-	A14	17.010.000.-
14	610	A16-A17	19.520.000.- – 21.350.000.-	A16	19.520.000.-
15	580	A18	23.200.000.-	A17	20.300.000.-
16	540	A18	21.600.000.-	A17	18.900.000.-
17	500	A19	22.500.000.-	A18	20.000.000.-
18	450	A19	20.250.000.-	A18	18.000.000.-
19	400	A20	20.000.000.-	A19	18.000.000.-
20	350	A20	17.500.000.-	A20	17.500.000.-
Total Clientes	11.620	Costo total de la campaña	372.000.000.-	Costo total de la campaña	262.660.000.-

Tabla N° 7.2: Comparativa asignación actual vs Método propuesto

Si bien el ahorro de tiempo no es muy significativo, por lo mismo no se agregó en esta tabla, sino que el potencial ahorro monetario al realizar la campaña mediante el modelo planteado alcanza aproximadamente los 90 millones de pesos, ahora es importante analizar cuál es el comportamiento de estos clientes luego de realizada las acciones obtenida con el método propuesto, dado que la aplicación de la acción pudiese no tener el impacto deseado, esto queda fuera del alcance de este proyecto, puesto que el análisis de esta información también conlleva una gran cantidad de trabajo.

Si bien el resultado de la acción arroja como resultado una mejora significativa del gasto, obviamente es imporante declarar que existirán clientes que no se encuentren conforme o que no se comporten del modo para lo cual la acción fue realizada, entonces como adelanto a esto se deja en claro que esos clientes serán tratado de una manera más particular, ahora de existir una cantidad muy amplia de clientes que supere los 1000 a lo largo de todas las unidades, se realizará una evaluación más exhaustiva del piloto ejecutado con anterioridad y declarado en este trabajo.

8 Conclusiones

Es claro que el Set Partitioning Problem puede aplicarse a muchos problemas de la vida real, lo cual permite basar la resolución de esos problemas en soluciones ya conocidas. Si bien en este problema puede resolverse a través de varios métodos en este proyecto en primera instancia se seleccionó la resolución mediante Constraint Programming y a través de la metaheurística Algoritmos Genéticos, y en comparación entre ambos seleccionar el que de mejor forma se comporta.

Es difícil imaginarse como las empresas obtienen información valiosa de todo el comportamiento que se realiza tanto en el consumo de productos o servicios de manera presencial como a través de dispositivos tecnológicos utilizando como canal la red, esto no es algo nuevo, sino que se viene utilizando ya hace años y seguirá cada vez haciéndose más fuerte buscando como fin entender el comportamiento de los clientes para poder prever el comportamiento de estos, lo cual permite generar mayores ingresos a los prestadores de servicios como a vendedores de productos específicos.

Con el conocimiento que se va adquiriendo a medida que se ha desarrollado el tema se abre una nueva visión del punto de vista de lo valioso que se transforma la información en el momento que se toma la decisión de manejarla y entenderla para lograr mediante distintos tipos de técnicas analizar ciertos puntos.

Si bien este proyecto se habla de dos técnicas para la resolución del Problema del Set Partitioning, esto representa una mínima parte de las técnicas existente para la resolución de este tipo de problemas. Las técnicas tomadas para realizar este trabajo fueron en base al conocimiento previo que se poseía de estas. Eso no quiere decir que en un futuro no se analizarán nuevas técnicas de resolución de la problemática planteada. Es claro que esto no fue considerado en este trabajo.

Este trabajo representa la ganancia de un nuevo conocimiento adquirido, y da un acercamiento a como se podrían resolver problemas de este tipo que antes de comenzar con el estudio no se tenía la claridad de la manera en que estos se resolvían.

En el ambiente y lugar que se trabaja diariamente es difícil encontrarse con estudios de este tipo, sobre todo para las empresas que apuntan siempre a generar ganancias mediante otros tipos de estudios, como por ejemplo de mercado, de competencias, etc. Pero, por otro lado existen estas técnicas que permiten realizar un tratamiento de la información de una manera distinta y que logran un buen resultado como se expresa en muchos casos en sitios y papers que se encuentran en internet, es interesante entonces plantear este problema y su resolución en el mundo real en el cual trabajamos diariamente y tener una visión un poco más atrevida y realizar este tipo de estudios que nos permitirá quizás minimizar los costos de la compañía, que es lo que se busca realmente.

Con la implementación de los algoritmos se llega a la conclusión de la importancia de las librerías que proveen terceros para minimizar el trabajo, y lograr algoritmos más robustos y encapsulables en un menor tiempo, si bien la mayoría de las librerías se deben adaptar al problema en particular este es un mínimo costo en comparación a desarrollar todo desde el comienzo.

Una vez realizado los experimentos, se pudo apreciar que para resolver el problema del Set Partitioning quizás la técnica de Algoritmos Genéticos no sea adecuada dado que las reparaciones a los cromosomas son muy costosas de ejecutar, teniendo en cuenta la cobertura que se debe realizar sin cubrir más de un grupo a la vez.

Se debe tener en cuenta siempre la cantidad y el cómo se ejecutan los experimentos, dado que estos pueden entregar resultados que en un futuro no llevan a tomar la mejor decisión, en primera instancia se realizó sólo una iteración de experimentos y llegando a una conclusión en base a esta iteración, pero una variable se dejó fuera y que es importantísima (el costo), una vez incorporada esta variable y realizada una nueva iteración de experimentos se tuvo la claridad para tomar una decisión con mayor respaldo.

Referencias

1.- Barceló Bugeda Jaume y Fernández Aréizaga Elena, *Métodos duales y algoritmos híbridos para problemas de "set partitioning", Trabajos de Investigación Operativa, Volumen: 5, págs. 35-59, 1990.*

2.- Balas Egon y Padberg Manfred, *Set partitioning: a survey, Management Sciences Research, Volumen 18, págs. 710-760, 1976.*

3.- Crawford Broderick , Soto Ricardo, Monfroy Eric, Castro Carlos, Palma Wenceslao y Paredes Fernando, *A Hybrid Soft Computing Approach for Subset Problems, Mathematical Problems in Engineering, Volume 2013, 2013.*

4.- Barber Federico y Salido Miguel. *Inteligencia Artificial: Técnicas, métodos y aplicaciones, Inteligencia Artificial, págs. 385-432, 2008.*

5.- Soto Ricardo, *A Gentle Introduction to CP, Constraint Programming, Curso MII 771, PUCV, 2014.*

6.- Marckworth Alan, *Consistency in networks of relations, Artificial Intelligence, Volumen 8, págs. 99-118, 1977.*

7.- Gaschnig Jonh, *A general backtrack algorithm that eliminates most redundant tests, Proceedings of the 5th international joint conference on Artificial intelligence, Volumen 1, págs. 457-457, 1977.*

8.- Manyá Felip y Gomes Carla, *Solution Techniques for Constraint Satisfaction Problems, Revista Iberoamericana de Inteligencia Artificial, Volumen 19, pags. 169-180, 2003.*

9.- Gaschnig Jonh, *Performance measurement and analysis of certain search algorithms, Technical report, Carnegie Mellon University, 1979.*

10.- Prosser Patrick, Hybrid algorithms for the constraint satisfaction problem, Computational Intelligence, Volumen 9, pags. 268-299, 1993.

11.- Haralick Robert y Gordon Elliot, *Increasing tree efficiency for constraint satisfaction problems, Artificial Intelligence, Volumen 14, págs. 263-314, 1980.*

12.- Dent Michael y Mercer Robert, *Minimal Forward Checking, 6th IEEE International Conference on Tools with Artificial Intelligence, págs. 432-438, 1994.*

13.- Van Roy Piter y Haridi Seif, *Concepts, Techniques, and Models of Computer Programming, The MIT Press, 2004, ISBN: 0262220695.*

14.- Melián Belén, Moreno Pérez José y Moreno Vega Marcos, *Metaheurísticas: una visión global, Revista Iberoamericana de Inteligencia Artificial, vol 19, págs. 7-28, 2003.*

15.- Golberg David, *Genetic Algorithms in Search, Optimization and Machine Learning, Addison-Wesley Longman Publishing Inc., 1989, ISBN: 0201157675.*

16.- Darwin Charles, *On the Origin of Species by Means of Natural Selection,* John Murray, 1859.

17.- Gestal Marcos y River Daniel, *Introducción a los algoritmos genéticos y la programación genética, Universidad da Coruña, 2010, ISBN: 9788497494229.*

18.- Moreno Julian, Rivera Juan Carlos y Ceballos Yoni, *Agrupamiento Homogéneo de elementos con múltiples atributos mediante Algorítmos Genéticos.* 2009, Dyna, Volumen 164, págs. 246 - 254, 2010.

19.- Jong, Kenneth, *An analysis of the behavior of a class of genetic adaptive systems, University of Michigan : PhD thesis, 1975.*

20.- Melián, Belén, Moreno, José, Moreno, Marcos. *Algoritmos Genéticos. Una visión práctica, Números: Revista de didactica de las matemáticas, Volumen 71, págs. 29 - 47,* 2009.

21.- Müller, Tobias, *Solving Set Partitioning Problems with Constraint Programmin, In Proceedings of the Sixth International Conference on the Practical Application of Prolog and the Fourth International Confernce on the Practical Application of Constraint Technology, págs. 313 -332, 1998.*

22.- Google, Librería OR-Tools. *https://code.google.com/p/or-tools/.* [En línea] 2012.

23.- Lapthorn, Barry, Librería btl.generic. *http://www.codeforge.com/article/180045.* [En línea] 2003.

24.- J E Beasley OR-Library, *http://people.brunel.ac.uk/~mastjjb/jeb/info.html.* [En línea] .