Análisis de estabilidad para problemas dinámicos de satisfacción de restricciones

Manuel Iván Angles Domínguez y Hugo Terashima Marín

Centro de Sistemas Inteligentes, ITESM Campus Monterrey
Ave. Eugenio Garza Sada 2501 Sur, C.P. 64849
Monterrey, Nuevo Leon., Mexico, Tel. +52 (81) 8328 4379
iangles@consultant.com, terashima@itesm.mx

Resumen Existen problemas de diferentes tipos que pueden ser modelados como problemas dinámicos de satisfacción de restricciones, dónde las restricciones, las variables o el dominio cambian con el tiempo. El objetivo al resolver estos problemas es disminuir el número de variables cuya asignación cambia entre problemas consecutivos, concepto conocido como distancia. En la actualidad este problema de estabilidad ha sido estudiado solamente para variaciones en las restricciones del problema. En este trabajo de investigación se realiza un amplio análisis del problema de estabilidad, al modificar las variables, dominios, restricciones y combinaciones de estos elementos para el problema de Asignación de Recursos. Se presentan resultados experimentales en cuanto a eficiencia, distancia y un nuevo parámetro denominado estabilidad global; para las técnicas de reuso de solución, reuso de razonamiento y una combinación de ambas. Además, se muestra que el comportamiento de la distancia es lineal con respecto a las variaciones.

1. Introducción

Un problema de satisfacción de restricciones (CSP por sus siglas en inglés, *Constraint Satisfaction Problem*) es un problema compuesto por un conjunto finito de variables, cada una de las cuales tiene asociado un dominio finito de valores y un conjunto de restricciones que limitan los valores que las variables pueden simultáneamente tomar. Formalmente [1] un problema de satisfacción de restricciones es una terna $C = (X, D, K)$, donde $X = \{x_1, \ldots, x_n\}$ es un conjunto de variables; $D = \{D_{x_1}, \ldots, D_{x_n}\}$ es un conjunto de dominios de las variables, uno para cada variable y K es un conjunto de restricciones, cada una de las cuales es un par $k = (Y, R)$, donde $Y = \{x_{i1}, \ldots, x_{ik}\}$ es el conjunto de variables que intervienen en la restricción y R es una relación sobre los dominios de estas variables, es decir, $R \subset D_{x_{i1}} \times \ldots \times D_{x_{ik}}$

Una solución a un CSP es una asignación de un valor de su dominio para cada variable, de tal forma que todas las restricciones son satisfechas al mismo tiempo.

Un DCSP [2] es una serie de CSP estáticos que cambian intermitentemente sobre el tiempo, debido a la evolución de sus componentes (variables, dominios

Gelbukh, M. Hernández Cruz (Eds.) Avances la

restricciones) dado cambios producidos por el ambiente (cambios en el conjunto de tareas a ser ejecutadas y/o de sus condiciones de ejecución, como en problemas de asiganación de recursos), un usuario (tratándose de un problema interactivo) u otro proceso (el DCSP sólo representa una parte de un problema más grande y otra entidad involucrada en el proceso de resolver el problema global, cambia su pensamiento de cómo el proceso de solución debería ser hecho; por ejemplo, un problema multiagente distribuido). En su trabajo original Dechter y Dechter [2] consideran únicamente modificaciones en las restricciones de los CSPs. En total, existen seis posibles alteraciones entre CSPs de un DCSP: adición o disminución de variables, adición o disminución de valores en dominios, adición o disminución de restricciones. Jonsson y Frank [1] presentan una definición formal que considera todas las posibles alteraciones:

> Sea $C = (X, D, K)$ un problema de satisfacción de restricciones. Cualquier problema de la forma $C' = (X', D', K')$ tal que $X' \supseteq X$ (i.e. hay más variables), $D'_x \subseteq D_x$ para cada $x \in X$ (i.e. hay menores valores en el dominio) y $K' \supseteq K$, (i.e. más limitaciones entre las variables) es una restricción de C. Un problema de la forma $C' = (X', D', K')$ tal que $X' \subseteq X$ (i.e. hay menos variables), $D'_x \supseteq D_x$ para cada $x \in X$ (i.e. hay más valores en los dominios) y $K' \subseteq K$, (i.e. menos limitaciones entre las variables) es una relajación de C. Un Problema Dinámico de Satisfacción de Restricciones es una secuencia de problemas de satisfacción de restricciones $C_0, C_1, \ldots$, tal que cada problema C_i es una restricción o relajación de C_{i-1}.

Solucionar un DCSP consiste en encontrar la solución de cada uno de los CSPs que lo conforman. Una forma de hacer esto es resolver cada CSP, uno tras otro desde el principio, enfoque que presenta dos defectos: ineficiencia y falta de estabilidad. En muchos casos algo del trabajo hecho en el previo CSP podría ser de alguna forma reusado para resolver el nuevo CSP, ya que para aplicaciones en tiempo real, el tiempo para resolver el nuevo problema es limitado. En cuanto a estabilidad, si la solución del previo CSP representa una asignación actualmente bajo ejecución, cualquier nueva solución debería minimizar el esfuerzo necesario para aplicar la reciente asignación. Para problemas interactivos descentralizados es preferible tener la menor cantidad de modificaciones a la actual solución.

La falta de estabilidad en las soluciones (conocido también como distancia entre soluciones) es un tópico de suma importancia, debido al costo en dinero y esfuerzo, que representa modificar la actual asignación, a fin de cumplir con los nuevos requerimientos; especialmente cuando típicamente no hay modelo del futuro, es decir, no hay ningún conocimiento acerca de cuándo y qué modificaciones se presentarán, lo que dificulta el desarrollo de soluciones que sean robustas a los cambios.

Para de evitar estos problemas actualmente existen dos enfoques para aprovechar alguna característica de un CSP para resolver el siguiente CSP que se produzca [3]. El primero de ellos, denominado reuso de solución, utiliza la solución previa (si hay alguna) para determinar la nueva solución. Verfaillie y

Schiex [4] desarrollaron un algoritmo basado en el enfoque de reuso de solució denominado *LocalChanges*, que esta inspirado en técnicas de *Backjumping* la variable actual viola una restricción se revisa la asignación de la variable co la que se tiene el conflicto, no la asignación de la variable inmediata anterior) de reparación iterativa [5] (consiste en utilizar una asignación previa y repararla usando una secuencia de modificaciones locales, es decir, modificaciones de sól una variable asignada).

El segundo enfoque, denominado reuso de razonamiento, trata de guarda de una manera concisa, una descripción aproximada de la frontera del es pacio de solución y justificaciones de esta frontera en términos del conjunto de restricciones. Schiex y Verfaillie [6] desarrollaron un método denominado *NogoodBuilder*, que consiste en ir generando a partir de las restricciones del CSP un conjunto de *nogoods*, definidos como instanciaciones parciales de vari ables que no pueden ser extendidas a una solución de todo el conjunto de vari ables. Estos *nogoods* pueden ser adicionados como restricciones al siguiente CSP generado.

Estos enfoques han sido examinados para variaciones de las restricciones de los CSPs, pero actualmente, no existe un análisis detallado de dichos algoritmos. El interés de este trabajo es analizar el desempeño de estos dos enfoques, con respecto a eficiencia y estabilidad, para modificaciones en las variables, dominio y restricciones de los CSPs, y además, proponer y analizar nuevas maneras de integrar estos enfoques, de tal manera que se obtenga un mejor desempeño que los algoritmos actuales.

Este artículo se encuentra organizado de la siguiente manera. En la sección 2, se establece la metodología que se siguió para realizar un análisis de estabil idad sobre los enfoques para resolver DCSP. En la sección 3 se presentan los resultados del análisis de estabilidad. Finalmente, las conclusiones del trabajo de investigación son presentadas en la sección 4.

2. Metodología

El primer paso consistió en modelar el problema de Asignación de Recursos (AR) como un DCSP. Posteriormente se desarrolló un generador aleatorio uniforme para crear los problemas de prueba de acuerdo al modelo de AR. A partir del análisis de los algoritmos basados en los enfoques de reuso de solución y reuso de razonamiento se desarrollaron dos nuevos algoritmos que integran los dos reusos en el mismo procedimiento. Entonces el análisis de estabilidad se realizó sobre los cuatro algoritmos y consistió principalmente en dos subanálisis: un análisis de sensibilidad y un análisis de distancia.

2.1. El problema de asignación de recursos

La asignación de recursos, es conocida por ser un problema difícil de gran importancia práctica, ya que existen un amplio rango de dominios que pueden ser modelados como problemas AR (telecomunicaciones satélitales [7], ruteo dinámico en

redes computacionales [8]). El problema de asignación de recursos [9] consiste en destinar recursos a un conjunto de tareas, programadas en intervalos de tiempo dados, de tal forma que ningún recurso es asignado a dos diferentes tareas al mismo tiempo. Este problema, así como los CSPs y DCSP son NP completos 0], lo que significa que para dimensiones reales no pueden ser óptimamente resueltos en una razonable cantidad de tiempo. La complejidad de cualquier procedimiento para resolver el peor caso posible para estos tipos de problemas es exponencial. Formalizando el modelo tenemos: Variables: $\{x_1, \ldots, x_n\}$; donde x_i representa la tarea i de un total de n tareas. Dominios: $D_{x_i} = \{R_1, \ldots, R_m\}$; donde R_j representa el recurso j de un total de m recursos que a la tarea x_i se puede asignar. Restricciones: $x_i \neq x_j$ si $i \neq j$; establece que las tareas x_i y $_j$ no pueden ocupar el mismo recurso, debido a que las dos se están llevando a cabo simultáneamente. Con el fin de extender este modelo y que los resultados puedan ser aplicados a un mayor número de dominios se consideraron otros dos pos de restricciones: $>$ y $<$. Por lo tanto, el problema de asignación de recursos modelado como DCSP, consiste en una serie de CSPs definidos por el modelo anterior, los cuales sufren modificaciones en sus variables, dominios o restricciones través del tiempo.

Se desarrolló un generador de DCSP, que crea instancias aleatorias uniformes de CSPs binarios, basados en las características del problema de asignación de recursos. El generador está habilitado para aumentar o disminuir entre CSPs número de variables, el tamaño del dominio, el número de restricciones y cualquier combinación de estos tres conceptos, ya que cada uno de ellos es manejado independientemente. El tamaño de dominio más pequeño que puede tener una variable es de 2 elementos. Tal valor fue elegido con el fin de no tener demasiado restringido los dominios y permitir un mayor trabajo de los algoritmos. Los valores que pueden tomar van de 1 al tamaño máximo de dominio especificado.

2. Algoritmos analizados

unto a los algoritmos *LocalChanges* [4] y *NogoodBuilder* [6], se desarrollaron y analizaron dos nuevos algoritmos. El primero de ellos, denominado *Lopt* está basado en *LocalChanges* y junto con un método para generar *nogoods* implícito en la técnica de reparación iterativa, hace reuso de razonamiento también. Su método para generar *nogoods* poda más espacio de búsqueda que el de *NogoodBuilder*. El segundo algoritmo desarrollado, denominado *Nopt*, complementa a *NogoodBuilder* haciendo reuso de solución, aprovechando la naturaleza recursiva del algoritmo. Estos dos nuevos algoritmos utilizan el reuso de solución cuando se tienen relajaciones en el DCSP y el reuso de razonamiento en el caso de restricciones, permitiéndoles evitar iniciar desde cero la solución de cualquier CSP del DCSP.

3. Análisis de sensibilidad y análisis de distancia

La sensibilidad de los algoritmos fue medida con base en los siguientes tres parámetros:

a) Estabilidad (Distancia): Distancia entre dos soluciones sucesivas, es decir el número de variables con diferente asignación entre soluciones de 2 CSPs contiguos.
b) Generación nula: Número de veces que la distancia entre CSPs contiguos fue cero. Actualmente, el concepto de estabilidad considera únicamente el número de variables cuya asignación varía entre soluciones de dos CSPs contiguos, sin importar cuántas veces dicha variación se presenta durante la solución completa del DCSP. Por eso, junto a la evaluación de la estabilidad se propone una nueva métrica para medir su frecuencia durante la resolución del DCSP.
c) Verificaciones de consistencia: Número de veces en que un algoritmo determina si un valor potencial para una variable viola alguna restricción con alguna otra variable.

Para tener una perspectiva más amplia acerca del desempeño de los algoritmos y especializarlos de acuerdo al tipo de variaciones que puede tener un DCSP, su sensibilidad a los cambios fue evaluada para las siguientes modificaciones en el DCSP: número de variables (V), tamaño de los dominios (D), número de restricciones (R), número de variables y del tamaño del dominio (V,D), número de restricciones y del tamaño del dominio (R,D), número de variables y del número de restricciones (V,R), número de variables, número de restriccicones y tamaño del dominio (V,R,D). Considerando que en la práctica la mayoría de los dominios sufren cambios lentamente y sólo en ocasiones especiales los cambios pueden salirse del promedio, se estableció que el máximo número de cambios (número de variables o restricciones que aumentan o disminuyen, número de variables de dominio será modificado y número de elementos que son adicionados al dominio) entre un CSP y otro es de tres.

Con base en que los cambios entre los CSP's son aleatorios y uniformes se estableció un conjunto de 30 DCSP de prueba para cada tipo de modificación. Cada DCSP está formado por 20 CSPs binarios creados por el generador aleatorio uniforme de DCSP. El CSP inicial está formado por 40 variables, tamaño máximo de dominio igual a 5 y 14 restricciones. El número de variables se cogió para dar una mayor amplitud a la variación de este concepto, de acuerdo al máximo número de cambios permitidos, que es de 3. Como el generador uniforme, podemos considerar un cambio promedio de 2 variables y que durante 10 de los 20 CSP que conforman el DCSP se aumentan variables y durante otros 10 se disminuye. Con estas características se esta manejando una variación de variables de alrededor del 50 % de las variables iniciales. El tamaño máximo de dominio fue escogido arbitrariamente para que con base en él se pudieran cular el número de restricciones. Un número de 14 restricciones fue determinado experimentalmente, tratando de obtener aproximadamente un 50 % de solubilidad en el DCSP, es decir, que alrededor de 10 de los 20 CSPs tuvieran solución. Experimentalmente se observó que el tiempo de resolución de los problemas varíaba entre 0 y 850 milisegundos. Con base en esto se estableció que los algoritmos tuvieran un tiempo límite para resolver el CSP_i que varía aleatoriamente entre 1 y 1000 milisegundos antes de pasar a resolver el siguiente CSP_{i+1}.

El objetivo del Análisis de Distancia es determinar la forma en que varía la distancia entre dos CSPs. Se consideró el caso de restricciones entre CSPs, ya que como se mencionó en la sección 2.3.1, en caso de relajación, simplemente se utiliza la solución del anterior CSP, por lo que la distancia entre soluciones es cero. La distancia fue evaluada para las siguientes modificaciones en el DCSP: aumento en el número de variables en 2, 4, 6, 8, 10, 12, 14, 16, 18 y 20 variables. Aumento en el número de variables cuyo dominio será modificado en 2, 4, 6, 8, 10, 12, 14, 16, 18 y 20 variables. El número de valores que fueron disminuidos del dominio fué seleccionado aleatoriamente para cada variable. Aumento en el número de restricicones en 1, 2, 3, 4, 5, 6, 7 restricciones.

Se eligieron estas modificaciones para abarcar una variación total del 50% de los parámetros iniciales. Para cada una de las posibles modificaciones mencionadas anteriormente se evaluaron 300 DCSPs modelando el problema de Asignación de Recursos, lo que significa que las restricciones manejadas fueron: $>, <, \neq$. Cada DCSP estaba formado por 2 CSPs. El CSP inicial está formado por 40 variables, tamaño máximo de dominio igual a 5 y 14 restricciones. Las razones para la elección de estos parámetros se describen en el análisis de sensibilidad.

Con el fin de analizar cómo afectaba a la distancia el hecho de utilizar los dos enfoques para resolver DCSP con repecto a emplear uno solo, el análisis se aplicó a los algoritmos *Lopt* (que integra reuso de solución y de razonamiento) y *Localchanges* (solo emplea reuso de solución). No se consideró tiempo límite para la solución de los CSPs.

3. Resultados y discusión

En el análisis de sensibilidad se obtuvo el comportamiento de los algoritmos en cuanto a eficiencia, estabilidad (tambien conocida como distancia) y generación nula; al variar las variables, dominios, restricciones y combinaciones de estos conceptos. El segundo grupo de experimentos fue enfocado a un análisis de distancia que se realizó para determinar si existe alguna relación que permita calcular el número de asignaciones que cambiarán entre soluciones de 2 CSPs.

3.1. Análisis de sensibilidad

En la comparación de los parámetros de medición del análisis de sensibilidad se analiza el promedio del desempeño de los algoritmos sobre el conjunto de los 30 DCSP. En el caso de la distancia entre soluciones, para cada DCSP, se considera un promedio sobre el número de veces que la distancia no fue nula.

Eficiencia.- En la figura 1 se grafican el número de verificaciones de consistencia que realizaron los algoritmos para cada una de las modificaciones analizadas.

En general se observa que *NogoodBuilder* y *Nopt* realizan un mayor número de verificaciones que *LocalChanges* y *Lopt*. Esto es debido al proceso de eliminación de consistencia de cada algoritmo, *Backjumping* para *NogoodBuilder*

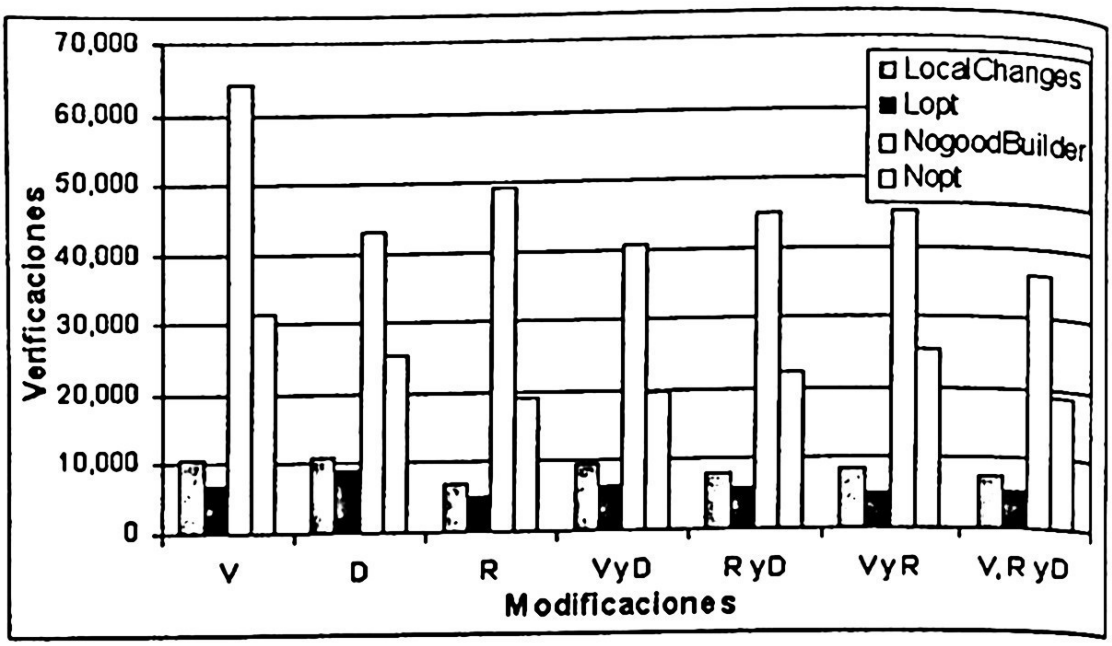

Figura 1. Resultados de eficiencia para el análisis de sensibilidad.

y *Nopt* y Reparación Iterativa para *LocalChanges* y *Lopt*, siendo este últim el que hace menos verificaciones de consistencia tal y como se observa en figura 1. Parte del ahorro de verificaciones es debido también a los enfoque de Reuso de Solución (RS) y Reuso de Razonamiento (RR). La diferencia entr *NogoodBuilder* y *Nopt* representa el trabajo que *Nopt* se ahorra por utilzar RS; es de aproximadamente el 50% de las verificaciones hechas por *NogoodBuilde Lopt* obtiene un ahorro de 30% sobre el trabajo hecho por *LocalChanges* debid al uso de RZ. El mayor ahorro que se presenta cuando se utiliza RR es debido que RZ solo poda ciertas ramas del árbol de búsqueda, mientras que RR elimin prácticamente todo el árbol cuando se tienen relajaciones entre los CSPs DCSP.

Distancia.- En la figura 2 se gráfican la distancia promedio que obtuviero los algoritmos para cada una de las modificaciones analizadas. En general observa que *LocalChanges* y *Lopt* generan una mayor distancia entre solucione contiguas. Esto puede deberse al proceso de búsqueda de cada algoritmo, para *Lopt* y *LocalChanges* es Reparación Iterativa (RI) y para *NogoodBuilde* es *Backjumping*. Entre *LocalChanges* y *Lopt* existe una pequeña diferencia cuál es producida por el reuso de razonamiento por parte de *Lopt*, ya que uso de *nogoods* implica no tomar en cuenta algunos valores de los dominios que obliga a asignar otros valores a las variables, aumentando la distancia entr las soluciones. En cambio, el reuso de solución que *Nopt* utiliza no afecta desempeño con respecto a *NogoodBuilder*, ya que este reuso evita precisamen que exista distancia entre las soluciones.

Generación Nula.- En la figura 3 se gráfican el número de CSPs cuy solución no sufrió modificaciones con respecto a la solución anterior. Entr *LocalChanges* y *Lopt* existe una pequeña diferencia la cuál es producida el reuso de razonamiento por parte de *Lopt*. Supongamos que el CSP_i no tie

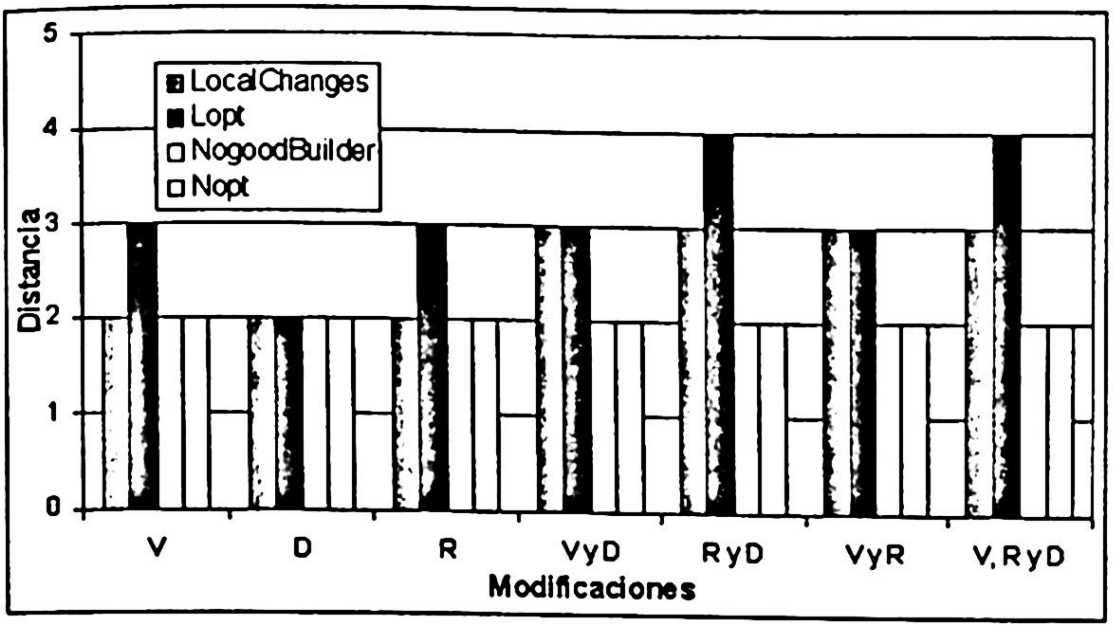

Figura 2. Resultados de distancia para el análisis de sensibilidad.

solución, pero se genera una instanciación parcial de variables de acuerdo al trabajo de procesamiento que llevaba el algoritmo antes de encontrar la falla. Para *LocalChanges* en CSP_{i+1} la nueva modificación puede permitir al algoritmo dar nuevos valores a las asignaciones antes de llegar a la inconsistencia. Al comparalas se encuentra que son diferentes disminuyendo la generación nula. En cambio, *Lopt* genera *nogoods* que le permiten desde el inico de solución de CSP_{i+1} determinar que no existe solución, por lo que no permite procesamiento que permita modificación de variables y se mantiene la generación nula. En promedio *Lopt* obtiene un aumento de 17% en la generación nula con respecto a *LocalChanges* y *Nopt* alrededor de un 14%, con respecto a *NogoodBuilder*, debido al reuso de solución.

El análisis de sensibilidad demostró que la utilización de integrar los enfoques de reuso de solución y reuso de razonamiento producen un mayor desempeño que utilizar un solo enfoque. Puesto que reuso de solución elimina verificaciones de consistencia y produce distancias cero entre soluciones aumentando la generación nula, es un enfoque que cualquier algoritmo para resolver DCSP debería utilizar. Esto conduce a enfocarnos a los casos de restricción de CSPs, en donde un algoritmo que utilice *nogoods*, con un proceso de reparación iterativa lo mas sistemático posible sería la mejor opción. Una propuesta para lograr esto es ajustar Reparación Iterativa para que empieze buscando soluciones cercanas a la establecida por una guía de *nogoods* y de ahí vaya ampliando su alcance de búsqueda. Los resultados obtenidos son independientes del tamaño del DCSP (en este caso fue de 20 CSPs), ya que las mediciones se hacen sobre CSPs contiguos, sin importar cuantos CSPs conforman el DCSP. Por otra parte, consideremos un DCSP cuyo CSP inicial tiene 100 variables. El comportamiento de los algoritmos al iniciar el CSP de 100 variables sería el mismo que si se considerara a este CSP como producto de otro DCSP que originalmente se inició con 40 variables (como

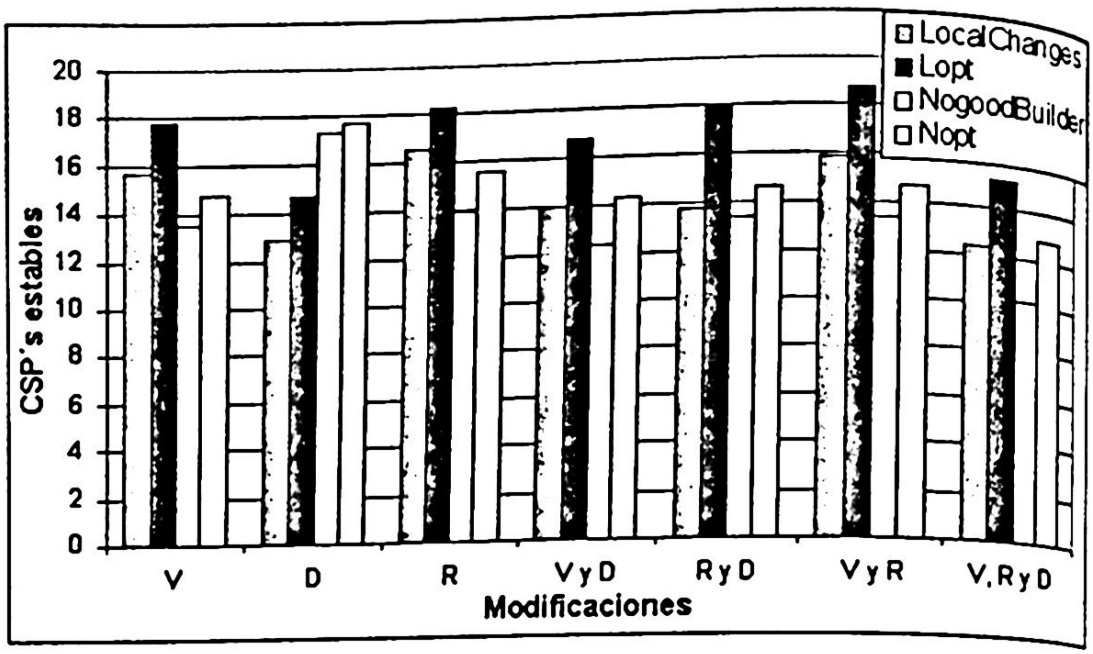

Figura 3. Resultados de generación nula para el análisis de sensibilidad

los examinados en este trabajo) y que ha sufrido modificaciones hasta llegar a variables. Por lo tanto, para CSPs con mayor número de variables se mantiene los resultados obtenidos en este trabajo. Un análisis semejante puede aplicar al tamaño del dominio y número de restricciones.

3.2. Análisis de distancia

En la figura 4 se muestra como aumenta la distancia al aumentar el núme de variables. Por ejemplo, si se aumenta el número de variables en un 40%, tiene un aumento de 20% en la distancia. De esta forma, para un CSP de variables, si una modificación produce un nuevo CSP con 16 nuevas variables, promedio, podría generarse una distancia de 8 variables con distinto valor entr las soluciones.

En la figura 5 se presenta el comportamiento de la distancia al variar número de variables a las que se les modifico su dominio. En este caso se obtie un mayor aumento en la distancia que el producido para la variación de variable En el caso de la figura 6, se ilustra la relación del incremento en la distancia modificar el número de restricciones.

Con base en las figuras 4, 5 y 6; se puede concluir que en general, el com portamiento de la distancia es lineal para los tres tipos de variaciones que analizaron. El análisis de distancia para restricciones en DCSP, demuestra la distancia entre soluciones varía de forma lineal con respecto a variacione en las variables, dominios y restricciones. Esta es la primera prueba de que posible determinar con base al número de elementos que varían, cual será la tancia que se produciría entre soluciones. Este comportamiento está acotado problema de Asignación de Recursos, por lo que futuros análisis para otros prob lemas modelados como DCSP y otros algoritmos son necesarios para verificar

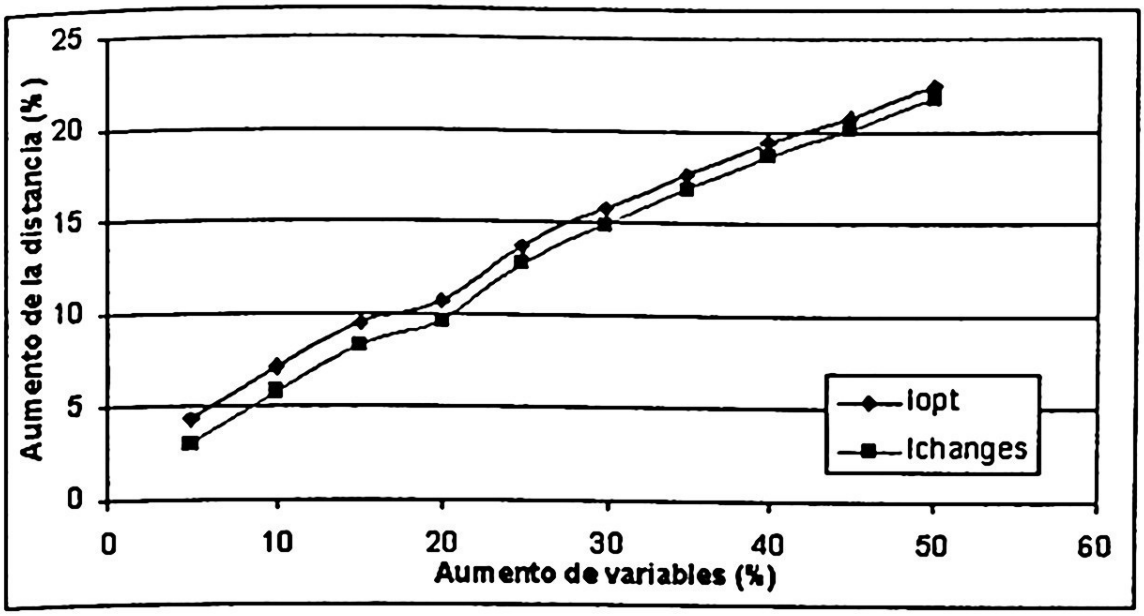

Figura 4. Comportamiento de la distancia al variar el número de variables.

la relación es siempre válida o bajo qué condiciones varía. El hecho de que para *LocalChanges* la relación sea lineal, implica que cualquier algoritmo que mejore *LocalChanges* tendrá que generar una relación con menor pendiente o en su mejor caso, una relación constante. Por otra parte, esta relación empírica podría ser utilizada como una heurística que permita minimizar la estabilidad local utilizando un algoritmo A* por ejemplo, el cuál minimiza una función objetivo utilizando información a través de una heurística.

4. Conclusiones

En este trabajo de investigación se desarrolló un análisis de estabilidad sobre cuatro algoritmos para resolver DCSP. Las pruebas realizadas sobre DCSP aleatorios han mostrado que ningún algoritmo obtiene el mejor desempeño en las tres medidas de comparación. En el caso de eficiencia, la técnica de reparación iterativa (*Lopt* y *LocalChanges*) obtiene un mejor desempeño que la ténica de *backjumping* de *NogoodBuilder* y *Nopt*. Por el contrario, *backjumping* obtiene menores distancias entre soluciones que reparación iterativa. En especial se ha observado que las características deseables en un algoritmo para resolver DCSP son: reuso de solución para el caso de relajaciones y reuso de razonamiento con un proceso de reparación iterativa sistemático para los casos de restricciones. Se ha verificado también la importancia del parámetro de medición propuesto, ya que los resultados indican que un algoritmo que produce buena estabilidad en el DCSP, no siempre producirá menor distancia entre soluciones. De esta manera, la generación nula permite elegir, de acuerdo al dominio de aplicación, que algoritmo es más conveniente de acuerdo a si se requiere mayor estabilidad durante todo el proceso de solución o menor estabilidad en pocas ocasiones. Con base en análisis de distancia se ha comprobado que existe una relación lineal entre el

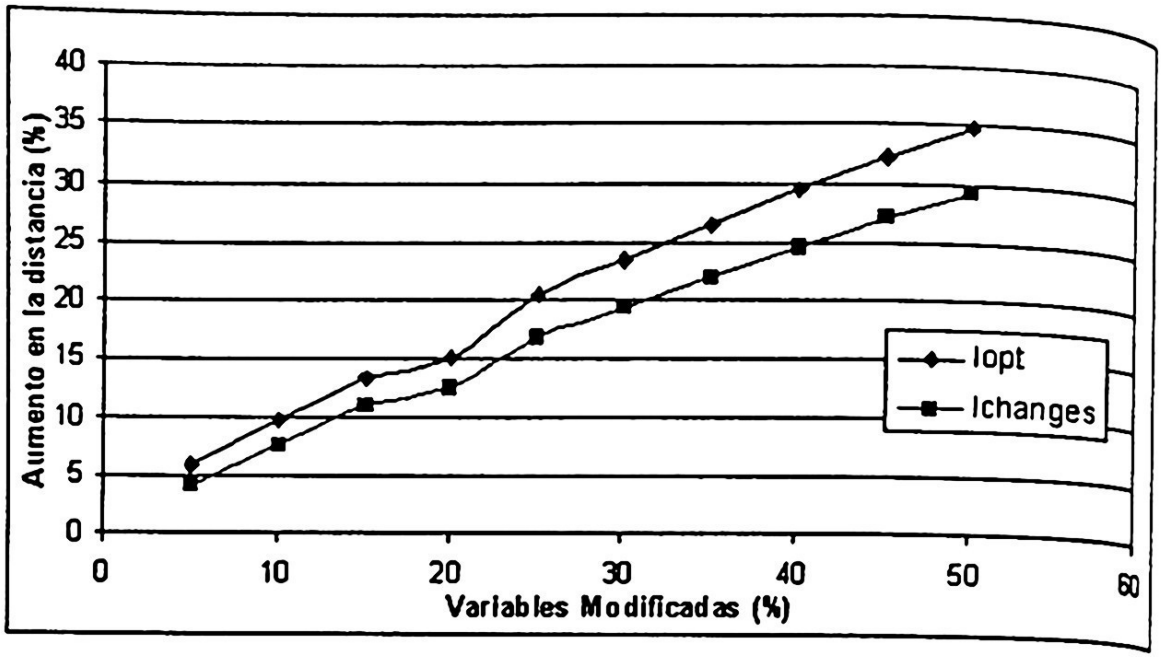

Figura 5. Comportamiento de la distancia al variar el tamaño del dominio.

incremento en la distancia de las soluciones con respecto a la variación individual del número de variables, tamaño de dominio y número de restricciones. La relación encontrada permite establecer, en forma promedio, el comportamiento de la distancia entre soluciones, lo que coadyuvará a una mejor toma de decisiones, en el caso de que las modificaciones puedan ser realizadas voluntariamente. El análisis de los algoritmos se realizó con el problema de asignación de recursos en su forma más general, es decir, sin especificar ningún dominio en particular; por lo que los resultados obtenidos pueden ser aplicados para cualquier dominio cuyo problema pueda ser presentado como asignación de recursos o en su caso a problemas que puedan ser modelados como CSPs binarios con las siguientes restricciones: $>, <, =$. Posibles extensiones a este trabajo incluyen realizar análisis de sensibilidad con otro tipos de problemas, que manejen otro tipo de restricciones o tamaño de dominio. Analizar si la relación de distancia se mantiene para otros tipos de problemas. Determinar límites inferiores o superiores al comportamiento de la distancia, así como desarrollar una teoría analítica de tal comportamiento. Este tipo de cuestiones requerirán respuesta con análisis más extensos en cuanto a formas del problema y tipos de modificaciones.

Agradecimientos

El primera autor agradece los apoyos recibidos del ITESM por la beca de Excelencia otorgada, del Centro de Sistemas Inteligentes del ITESM y de la Cátedra de Investigación CAT010.

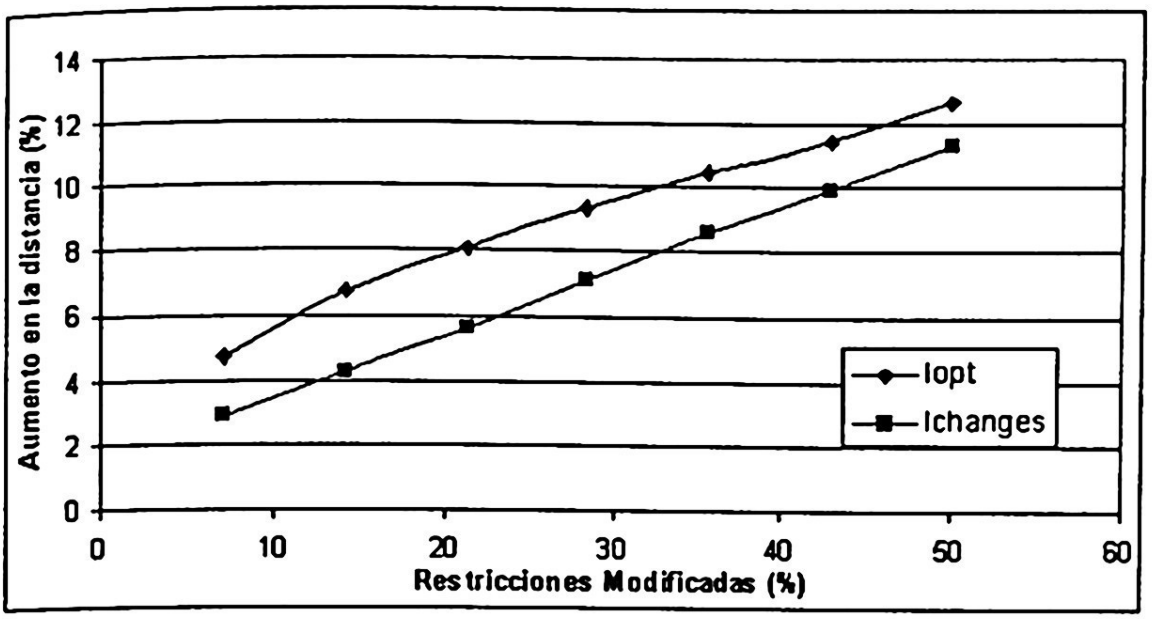

Figura 6. Comportamiento de la distancia al variar el número de restricciones.

Referencias

1. A.K. Jonsson and J. Frank. A framework for dynamic constraint reasoning using procedural constraints. In *European Artificial Intelligence Conference*, 2000.
R. Dechter and A. Dechter. Belief maintenance in dynamic constraint networks. In *Proceedings of the Sixth National Conference on Artificial Intelligence*, pages 37–42, 1988.
T. Schiex and G. Verfaillie. Two approaches to the solution maintenance problem in dynamic constraint satisfaction problems. 1993.
G. Verfaillie and T. Schiex. Solution reuse in dynamic constraint satisfaction problems. In *AAAI, Vol. 1*, pages 307–312, 1994.
S. Minton, M. D. Johnston, A. B. Philips, and P. Laird. Minimizing conflicts: A heuristic repair method for constraint satisfaction and scheduling problems. *Artificial Intelligence*, 58(1-3):161–205, 1992.
T. Schiex and G. Verfaillie. Nogood recording for static and dynamic constraint satisfaction problems. 1993.
. C. Plaunt, A.K. Jonsson, and J. Frank. Run-time satellite tele-communications call handling as dynamic constraint satisfaction. In *Proceedings of the IEEE Aerospace Conference 1999*, pages 165–174, 1999.
Murat Alanyali and Bruce Hajek. On simple algorithms for dynamic load balancing. In *INFOCOM (1)*, pages 230–238, 1995.
B. Y. Choueiry, G. Noubir, and B. Faltings. Blending AI and mathematics: the case of resource allocation. In *Fourth International Symposium on Artificial Intelligence and Mathematics*, pages 32-37, Fort Lauderdale, Florida, 1996.
C. P. Gomes and J. Hsu. ABA: An assignment based algorithm for resource allocation. *SIGART Bulletin*, 7(1):2–8, 1996.