Uso de Deducciones de no Viabilidad en el Cálculo de Arco-Consistencia

Frank D. Valencia.
e-mail: fvalenci@sofia.univalle.edu.co

Camilo Rueda.
e-mail: crueda@borabora.univalle.edu.co

Grupo AVISPA
Pontificia Universidad Javeriana de Cali
Dirección: Calle 18 Num. 118-250 Vía Pance
Teléfono: (57) 2-5552175
Fax: (57) 2-5552823
Cali, Colombia.

Resumen

Los problemas de Satisfacción de Restricciones, conocidos como CSP ("Constraint Satisfaction Problems") constituyen actualmente una de las áreas centrales de la Inteligencia Artificial. Las técnicas de Arco-Consistencia (AC) se utilizan ampliamente en el tratamiento de CSP. Recientemente se han propuesto para el cálculo de Arco-Consistencia los algoritmos AC6++ y AC-7, que mantienen las denominadas cuatro propiedades deseables de un algoritmo de Arco-Consistencia y con base en ellas se afirma su optimalidad. Se muestra en este artículo que dichas propiedades pueden no ser suficientes. Se presenta una nueva propiedad deseable, independiente de la semántica de las restricciones, no considerada anteriormente por ningún algoritmo de AC. Esta propiedad es *necesaria* para garantizar algoritmos óptimos. Su efectividad se muestra analítica y experimentalmente modificando AC6++ y AC-7.

1. INTRODUCCIÓN

El concepto de problema de satisfacción de restricciones (CSP) constituye hoy un área fundamental de la inteligencia artificial. Esta importancia se debe a que bajo este concepto pueden formularse muchos problemas interesantes de carácter combinatorio. Un CSP se define por un conjunto de variables sujetas a un conjunto de restricciones y un conjunto finito (dominio) de valores asociado a cada variable. Una solución a un CSP, es una asignación de valores a variables que satisfacen todas las restricciones.

Las técnicas de Consistencia, especialmente las de Arco-Consistencia (AC) han sido ampliamente estudiadas para simplificar el CSP antes o durante la búsqueda de soluciones. Sabin [7] ha mostrado experimentalmente la importancia de usar AC durante la búsqueda de soluciones.

Varios algoritmos de AC han sido propuestos, comenzando en 1972 con el algoritmo de Waltz [11] y terminando recientemente con los óptimos AC-6++ [3] y AC-7 [2]. Entre los algoritmos de AC, son de especial interés: AC-3 [5], AC-4 [6], AC-5 [9], AC-6 [1], AC-*Inference* [4], AC-6++ y AC-7. AC-3 es el modelo de los restantes. AC-4 introduce estructuras de datos para un procesamiento dinámico. AC-5 es un algoritmo genérico que puede ser optimado para ciertas clases importantes de restricciones, o especializado para obtener AC-3 o AC-4. AC-6 es un algoritmo incremental basado en AC-4. AC-*Inference* es un algoritmo que al igual que AC-5 puede tomar ventajas de las propiedades de las restricciones. AC-6++ y AC-7 refinan AC-6, manteniendo *las cuatro propiedades deseables* de un algoritmo de Arco-Consistencia.

El estándar de comparación en los algoritmos de AC lo constituyen el número de chequeos de restricciones realizados. Con base en este estándar, tres de las cuatro propiedades deseables establecen condiciones para reducir el número de chequeos de restricciones. La cuarta propiedad establece una condición sobre la complejidad en espacio.

En este trabajo se presenta una nueva propiedad deseable, no considerada por ningún algoritmo de AC previo, que ofrece una reducción en el número de chequeos de restricciones, como se muestra analítica y experimentalmente al modificar AC-7 y AC6++. Estas modificaciones serán denominadas AC6-3+ y AC-7+, respectivamente.

Este artículo está organizado así: En la sección 2 se presentan definiciones generales; en la sección 3 se tratan las cuatro propiedades deseables; en la sección 4 se presenta la nueva propiedad deseable, una descripción genérica de un algoritmo que la considera y su análisis; en la sección 5, se muestran resultados experimentales. Las conclusiones se presentan en la sección 6.

2. PRELIMINARES

Un CSP se define como una tripla $<V,D,C>$. $V=\{x_1,...,x_n\}$ es el conjunto de variables, con su domino de valores asociado en $D=\{D_1,...,D_n\}$, donde el dominio de la variable x_i es D_i. C es un conjunto de restricciones $R_{(i_1,...,i_u)}$ que denota combinaciones de valores permitidas para las variables $x_{i_1},...,x_{i_u}$ de V, (i.e., $R_{(i_1,...,i_u)} \subseteq D_{i_1} \times ... \times D_{i_u}$). Utilizamos el predicado $R_{(i_1,...,i_u)}(v_{i_1},...,v_{i_u}) \Leftrightarrow (v_{i_1},...,v_{i_u}) \in R_{(i_1,...,i_u)}$. Se dice que un CSP es *binario* si todas las restricciones en C son binarias, es decir, de la forma $R_{(i,j)}$. Por claridad en este trabajo se trata el caso binario, sin embargo la generalización a n-ario es inmediata.

Calcular Arco-Consistencia significa eliminar algunos valores de los dominios, que no pueden aparecer en ninguna solución. Estos valores se dicen no *viables*.

Definición 1. $w \in D_j$ es un *soporte* para $v \in D_i$ en $R_{(i,j)}$ ssí $R_{(i,j)}(v,w)$. $v \in D_i$ es *viable con respecto a* $R_{(i,j)}$ ssí $\exists w \in D_j$: w es un soporte para v en $R_{(i,j)}$. $v \in D_i$ es *viable* ssí $\forall R_{(i,j)} \in C$: v es *viable con respecto a* $R_{(i,j)}$.

Los algoritmos de AC operan sobre un grafo de restricciones que representa al CSP. Los nodos de este grafo corresponden a las variables y los arcos a las restricciones. Por cada restricción $R_{(i,j)}$, el grafo tiene dos arcos (i,j) y (j,i). El arco (i,j) es asociado a $R_{(i,j)}$ y el arco (j,i) a $R_{(j,i)}$, que es igual $R_{(i,j)}$ excepto que sus argumentos están intercambiados (i.e., $R_{(j,i)}(w,v)=R_{(i,j)}(v,w)$). Se dice que $R_{(i,j)}$ y $R_{(j,i)}$ cumplen *la propiedad de bidireccionalidad.* Sin pérdida de generalidad, se asume que sobre un mismo par de variables existe a lo sumo una restricción. Entonces, Un CSP binario $<V,D,C>$ se representa con el grafo de restricciones $G=<V,A>$ donde $A=\{ (x_i,x_j) \mid R_{(i,j)} \in C \vee R_{(j,i)} \in C\}$. En este trabajo cada arco (x_i,x_j) en A se abrevia como (i,j). Denotamos $Arcos(G)=A$, y $Nodos(G)=V$.

La siguientes definiciones de uso general describen Arco-Consistencia en un arco y en un grafo [9]:

Definición 2. Sea $(i,j) \in Arcos(G)$. El arco (i,j) es *Arco-Consistente con respecto a* D_i y D_j ssí $\forall v \in D_i$:v es viable con respecto a $R_{(i,j)}$. Sea $\rho' = D_1 \times ... \times D_n$. G es *Arco-Consistente* con *respecto a* ρ' ssí $\forall (i,j) \in Arcos(G)$:$(i,j)$es Arco-Consistente con respecto a D_i y D_j. Sea $\rho = D_1 \times ... \times D_n$. Sea $\rho' \subseteq \rho$. G *es maximalmente arco-consistente con respecto a* ρ' *en* ρ ssí G es Arco-Consistente con respecto ρ' y no existe otro ρ'' con $\rho' \subset \rho''$ tal que G es Arco-Consistente con respecto ρ''.

Dado G y ρ, el propósito de un algoritmo de AC es calcular un ρ' tal que G sea maximalmente arco-consistente con respecto ρ' en ρ.

En el cálculo de Arco-Consistencia, se define un ordenamiento total "$\leq$" en los dominios.

Definición 3. El soporte inferior en D_j para un valor $v \in D_i$, es un valor $SI_{[(i,j),v]}$ tal que cualquier valor en D_j menor que $SI_{[(i,j),v]}$ no es compatible con v. La *propiedad del soporte inferior* establece: $\forall (i,j) \in Arcos(G)\left(\forall v \in D_i, \forall w \in D_j \left(w < SI_{[(i,j),v]} \Rightarrow \neg R_{(i,j)}(v,w)\right)\right)$. Note que $SI_{[(i,j),v]}$ no necesariamente es un soporte.

En las próxima sección se mostrará que las propiedades del soporte inferior y de bidireccionalidad, son bastante útiles en el Cálculo de Arco-Consistencia.

3. LAS CUATRO PROPIEDADES DESEABLES

El estado del arte en el cálculo de Arco-Consistencia reside en cuatro *propiedades deseables* que definen el comportamiento ideal de un algoritmo de AC [3]. Estas propiedades están orientadas a determinar un número de chequeos óptimo con una complejidad espacial práctica y pueden ser derivadas de las propiedades del soporte inferior y de la bidireccionalidad de las restricciones.

Las propiedades deseables de un algoritmo de arco-consistencia establecen que:

1. Nunca se debe chequear $R_{(i,j)}(v,w)$ si existe un w' en D_j tal que $R_{(i,j)}(v,w')$.
2. Nunca se debe chequear $R_{(i,j)}(v,w)$ si existe un w' en D_j tal que $R_{(j,i)}(w',v)$.
3. Nunca se debe chequear $R_{(i,j)}(v,w)$ si:
   a. $R_{(i,j)}(v,w)$ ha sido chequeada previamente,
   o b. $R_{(i,j)}(v,w)$ ha sido chequeada antes.

4. Se debe tener una complejidad espacial $O(ed)$, donde $e=|C|$ y $d=\underset{1\leq i\leq n}{Máx}(|D_i|)$.

La propiedad 1, se puede mantener utilizando la propiedad del soporte inferior. La propiedad 2, utilizando soporte inferior y bidireccionalidad. La propiedad 3a, utilizando soporte inferior, la 3b utilizando soporte inferior y bidireccionalidad. La propiedad 4 es una consideración práctica sobre el tamaño de las estructuras de datos. AC-3 no mantiene las propiedades 1,2,3. AC-4 no mantiene 1,2,3b,4. AC-6 no mantiene 2 y 3b (las propiedades derivadas de la bidireccionalidad). AC-*Inference* no mantiene la propiedad 4. AC6++ y AC-7 mantienen la cuatros propiedades, y por esta razón son considerados óptimos en [3].

AC6++ y AC-7 solo difieren en la forma en que los valores se tratan. Mientras AC6++ se aproxima en esta forma a AC-6, AC-7 coincide en esto con AC-*Inference*. La idea de AC-7 (y de AC-*Inference*) consiste en propagar los efectos de eliminar un valor no viable (i.e., reconsiderar la viabilidad de los valores que éste soportaba) tan pronto como sea posible, con el fin de reducir chequeos de restricciones innecesarios y también de descubrir dominios vacíos rápidamente. En la práctica, en promedio, AC-7 ha demostrado ser superior a AC6++ al utilizar ésta idea.

En [2] se afirma que con estas propiedades AC6++ garantiza un número de chequeos óptimo dado un ordenamiento de valores, de variables y de arcos (el orden en que los valores, variables y arcos son considerados para calcular Arco-Consistencia). Sin embargo es posible encontrar casos excepcionales en los cuales AC-6 puede chequear considerablemente menos restricciones que AC6++. Son casos en los que es imposible establecer el mismo ordenamiento de valores en AC-6 y AC6++ [8].

Las anteriores propiedades son también importantes como heurísticos para reducir el chequeo de restricciones, para garantizar que un chequeo no sea realizado más de una vez, y para no realizar chequeos innecesarios.

4. LA NUEVA PROPIEDAD DESEABLE

La eliminación de un valor de un dominio impone la reconsideración de la viabilidad de valores en otros dominios para los cuales éste era soporte. Siguiendo la idea de AC-7, si se desea propagar las

consecuencias de la eliminación de un valor tan pronto ésta como suceda, es razonable pensar que se deberían realizar estas eliminaciones tan pronto como sea posible.

La nueva propiedad deseable ayuda a encontrar valores no viables rápidamente, con base en el principio que denominamos deducciones *de no viabilidad.* Tales deducciones evitan, por una parte, que se busquen soportes de valores no viables y, por otra, que valores no viables se asignen como soportes de otros. Esta propiedad no es considerada por ningún algoritmo previo.

La quinta propiedad deseable establece que valores para los cuales se pueda deducir que no son viables no deben ser chequeados. Esta propiedad puede expresarse de la siguiente manera:

5. Nunca se debe chequear $R_{(i,j)}(v,w)$ cuando se pueda deducir que v o w no son viables.

AC-5 deduce la no viabilidad de un valor en clases especiales de restricciones, pero no en restricciones arbitrarias.

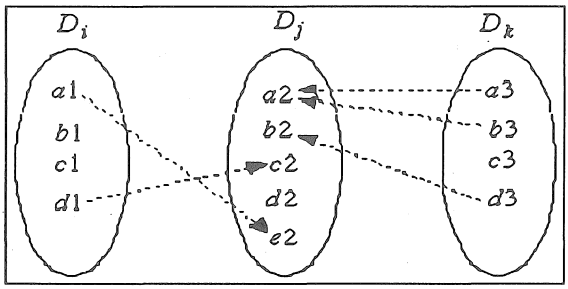

Figura 1. Una flecha de x a y, indica que y es el primer soporte de x.

Suponga que algún algoritmo de AC (usando un ordenamiento total en los dominios) ha chequeado todos los valores en D_i buscando soporte en D_j (ver Figura 1). Los valores $a1$ y $d1$ tienen a $e2$ y $c2$, respectivamente, como primer soporte en D_j. Los valores $b1$ y $c1$ al no tener soporte en D_j se eliminan. Seguidamente, todos los valores D_k se chequean buscando un soporte en D_j. El valor $c3$ es eliminado por no tener soporte en D_j, pero $a3$ y $b3$ tienen como soporte a $a2$, y $d3$ tiene a $b2$ como soporte. Sin embargo, se sabe que los valores $a2$ y $b2$ no son viables, pues todos los valores de D_i fueron chequeados y ninguno fue compatible con ellos (por el concepto de primer soporte y bidireccionalidad). No eliminar $a2$ y $b2$ cuando es posible deducir su no

viabilidad, puede producir propagaciones y chequeos innecesarios. No obstante, ninguno de los algoritmos previos evitan subsecuentes chequeos con $a2$ y $b2$.

Las cuatro propiedades deseables evitan chequear una pareja de valores cuando ésta ha sido anteriormente chequeada. La nueva propiedad propuesta evita chequear una pareja de valores cuando alguno de sus componentes se deduce no viable. Ahora se presentarán dos formas de deducciones de no viabilidad que son fácilmente adaptables en AC6++ y AC-7.

4.1 Propiedad 1 (Mínimo Soporte Inferior). Sea $MSI_{(i,j)} = \underset{v \in D_i}{Min}(SI_{[(i,j),v]})$ entonces $\forall w \in D_j \ (w < MSI_{(i,j)} \Rightarrow w \text{ no es viable})$.

Demostración : La propiedad del soporte inferior es equivalente a:

$\forall v \in D_i, \forall w \in D_j \ w < SI_{[(i,j),v]} \Rightarrow \forall v \in D_i, \forall w \in D_j : \neg R_{(i,j)}(v,w)$ (E.1).

Trivialmente, $\forall w \in D_j (\ w < MSI_{(i,j)} \Rightarrow \forall v \in D_i \ (w < SI_{[(i,j),v]}))$, que es equivalente a:

$\forall w \in D_j (\ w < MSI_{(i,j)}) \Rightarrow \forall w \in D_j, \forall v \in D_i \ (w < SI_{[(i,j),v]})$ (E.2).

De (E.2) y (E.1) se deduce: $\forall w \in D_j \ (w < MSI_{(i,j)} \Rightarrow \forall v \in D_i : \neg R_{(i,j)}(v,w))$. Por bidireccionalidad se tiene: $\forall w \in D_j (\ w < MSI_{(i,j)} \Rightarrow \forall v \in D_i : \neg R_{(j,i)}(w,v))$.

La propiedad establece que todos los valores en D_j menores que el mínimo soporte inferior de los valores en D_i, no son viables y por lo tanto pueden ser eliminados sin chequeos adicionales.

4.2 Propiedad 2 (Cero-Cardinalidad). Sea $PC_{[(i,j),v]} \geq \left| \{ v \in D_i | R_{(i,j)}(v,w) \} \right|$ una cota superior del número de soportes de $v \in D_i$ en D_j. Entonces $PC_{[(i,j),v]} = 0 \Rightarrow v$ no es viable.

4.3 Algoritmo. Aquí presentaremos un algoritmo genérico basado en AC-5 que considera las propiedades de Cero-Cardinalidad y la del Mínimo Soporte Inferior. Este algoritmo puede especializarse para obtener cualquiera de los algoritmos de AC.

Todos los algoritmos de AC usan una lista de espera Q. En este esquema genérico Q tiene elementos de la forma $<(i,j),w>$ donde (i,j) es un arco y w es un valor que ha sido borrado de D_j y justifica la necesidad de reconsiderar para Arco-Consistencia el arco (i,j) (i.e., para la propagación de los efectos de la eliminación de w). El procedimiento Encolar(i,Δ,Q) se usa cuando un conjunto Δ ha sido eliminado de D_i (Figura 2), para agregar a la cola Q, elementos de la forma

$<(k,i),v>$ donde (k,i) es un arco y $v \in \Delta$. En las especificaciones, un parámetro indexado con 0 (i.e., de la forma p_o) representa su valor en el momento de la invocación del procedimiento o función. El procedimiento Decolar (Q,i,j,w) saca un elemento $<(i,j),w>$ de la cola Q (Figura 2).

proc EnColar (**ent** i,Δ, **ent−sal** Q)
Pre: $\Delta \subseteq D_i \wedge x_i \in Nodos(G)$
Post: $Q = Qo \cup \{<(k,i),v> | (k,i) \in Arcos(G) \wedge v \in \Delta\}$.

proc DeColar (**ent−sal** Q, **sal** i,j,w)
Post: $<(i,j),w> \in Qo \wedge Q = Qo - \{<(i,j),w>\}$.

Figura 2. Operaciones sobre Colas.

proc ArcCons (**ent** i,j **sal**, Δ_i, Δ_j)
Pre: $(i,j) \in Arcos(G)$.
Post: $\Delta_i = \{v \in D_i | \forall w \in D_j : \neg R_{(i,j)}(v,w)\} \wedge$
$\Delta_j = \{w \in D_j | w < MSI_{(i,j)}\}$.
proc LocalArcCons (**ent** i,j,w **sal** Δ_i, Δ_j)
Pre: $(i,j) \in Arcos(G) \wedge w \notin D_j$.
Post:
$\Delta_i = \{v \in D_i | R_{(i,j)}(v,w) \wedge \forall w' \in D_j : \neg R_{(i,j)}(v,w')\}$
$\wedge \Delta_j = \{w \in D_j | w < MSI_{(i,j)}\}$.

proc ArcCons (**ent** i,j **sal**, Δ_i, Δ_j)
Pre: $(i,j) \in Arcos(G)$.
Post: $\Delta_i = \{v \in D_i | \forall w \in D_j : \neg R_{(i,j)}(v,w)\} \wedge$
$\Delta_j = \{w \in D_j | PC_{[(j,i),w]} = 0\}$.
proc LocalArcCons (**ent** i,j,w **sal** Δ_i, Δ_j)
Pre: $(i,j) \in Arcos(G) \wedge w \notin D_j$.
Post:
$\Delta_i = \{v \in D_i | R_{(i,j)}(v,w) \wedge \forall w' \in D_j : \neg R_{(i,j)}(v,w')\}$
$\wedge \Delta_j = \{w \in D_j | PC_{[(j,i),w]} = 0\}$.

Figura 3. Procedimientos ArcCons y LocalArcCons modificados para considerar la propiedad del Mínimo Soporte Inferior (izq.) y la propiedad Cero-Cardinalidad (Der.).

El procedimiento ArcCons(i,j,Δ_i,Δ_j) calcula el conjunto de valores $\Delta_i \subseteq D_i$ para la variable x_i que no tienen soporte en D_j y *deduce* el conjunto de valores $\Delta_j \subseteq D_j$ que no tienen soporte en D_i. El procedimiento LocalArcCons(i,j,w,Δ_i,Δ_j) se usa para calcular el conjunto de valores $\Delta_i \subseteq D_i$, que ya no tienen soporte en D_j por causa de la eliminación de w de D_j y *deduce* también el conjunto de valores $\Delta_j \subseteq D_j$ que ya no tienen soporte en D_i. En la Figura 3, se especifican éstos procedimientos, usando como deducción la propiedad del mínimo soporte inferior (Izq.) y la propiedad Cero-Cardinalidad (Der.). En ningún algoritmo propuesto anteriormente se contempla Δ_j.

El algoritmo genérico tiene dos fases (ver Figura 4). En la fase de inicialización (líneas 2-7), a cada arco se le aplica Arco-Consistencia. En la fase de propagación (líneas 8-14), se propagan los efectos de las eliminaciones de valores no viables. Todo valor eliminado se guarda en la lista de

espera Q, hasta que se propaguen los efectos de su eliminación. El algoritmo termina cuando no existen más elementos en Q .

```
Algoritmo AC.
```
Post: Sea $\rho_o = D_{1_o} \times ... \times D_{n_o} \wedge \rho_f = D_{1_f} \times ... \times D_{n_f}$.
G es máximalmente Arco-Consistente con respecto a ρ_f en ρ_o.

```
{
1  Q←∅;
2  Para cada (i,j)∈Arcos(G) haga
3     ArcCons(i,j,Δi,Δj);
4     Encolar(i,Δi,Q);
5     Di←Di−Δi;
6     Encolar(j,Δj,Q);
7     Dj←Dj−Δj;
8  Mientras Q≠∅ haga
9     Decolar(Q,i,j,w);
10    LocalArcCons(i,j,w,Δi,Δj);
11    Encolar(i,Δi,Q);
12    Di←Di−Δi;
13    Encolar(j,Δj,Q);
14    Dj←Dj−Δj;
}
```

Figura 4. Algoritmo genérico de Arco-Consistencia.

Modificamos AC6++ añadiendo la propiedad del soporte inferior. Denominamos AC6-3+ el algoritmo resultante. Después de calcular el conjunto $\Delta_i \subseteq D_i$ mencionado anteriormente, se calcula el Mínimo Primer Soporte (i.e., $MSI_{(i,j)}$) y se eliminan los valores previos a éste.

Modificamos también AC-7 añadiendo la propiedad Cero-Cardinalidad. El algoritmo resultante lo denominamos AC-7+. Para usar la propiedad Cero-Cardinalidad, inicialmente cada $PC_{[(i,j),v]}$ se iguala a $|D_j|$. $PC_{[(i,j),v]}$ se decrementa cuando una restricción $R_{(i,j)}(v,w)$ no es satisfecha, cuando un soporte w' para v es eliminado, o cuando se encuentra que alguno de los valores para los cuales v es asignado como soporte, es eliminado. Cuando $PC_{[(i,j),v]}=0$, v se elimina de D_i .

4.3.1 Análisis.

Aquí se presentan un esbozo de la prueba de correctitud del algoritmo genérico AC.

Sea $\rho_o = D_{1_o} \times ... \times D_{n_o}$ y $\rho_f = D_{1_f} \times ... \times D_{n_f}$ tal que G es maximalmente Arco-Consistente con respecto a ρ_f en ρ_o. ρ_f es el conjunto que se espera calcular. Sea D_{i_k} $(1 \le i \le n)$ el dominio actual de la variable x_i después de la eliminación de un conjunto de valores de $D_{j_{k-1}}$ $(1 \le j \le n)$ (i.e., de la $k-$ésima ejecución de una eliminación de valores). Si $k=0$, D_{i_k} es el dominio inicial de la variable x_i cuando el algoritmo comienza. Sea $\rho_k = D_{1_k} \times ... \times D_{n_k}$. (i,a) denota el valor $a \in D_i$.

Todo valor (i,a) es eliminado de D_{i_k} sólo si no es viable, sea por el mismo principio de los algoritmos previos, sea por las propiedades Cero-Cardinalidad y Mínimo Soporte Inferior. Si cada valor (j,b) eliminado previamente no está en D_{j_f}, entonces (i,a) no está en D_{i_f}. Por definición $\rho_f \subseteq \rho_o$ y $\rho_k \subseteq \rho_{k-1}$, luego $\rho_f \subseteq \rho_k$. Entonces el algoritmo mantiene el invariante $\rho_f \subseteq \rho_k \subseteq \rho_{k-1} \subseteq ... \subseteq \rho_o$. Cada vez que un valor (j,b) es eliminado, es puesto en Q hasta que los valores que éste era soporte busquen nuevos soportes. Cada vez que se encuentra un valor (i,a) sin soporte, es eliminado de su dominio actual. Por lo tanto después de la fase de inicialización, cada valor tiene un soporte en los dominios actuales o en $Q' = \{w | < (i,k), w) \in Q\}$. Entonces el algoritmo mantiene el invariante: $\forall (i,j) \in Arcos(G) (\forall v \in D_{i_k} (\exists w \in (D_{j_k} \cup Q') : R_{(i,j)}(v,w)))$. El algoritmo termina cuando $Q = \varnothing$, entonces por definición, G es arco-consistente con respecto a ρ_k en ρ_o y del primer invariante se tiene que $\rho_f \subseteq \rho_k$, luego G es maximalmente arco-consistente con respecto a ρ_k en ρ_o (i.e. $\rho_f = \rho_k$) cuando el algoritmo termina.

Siendo extensiones de AC6++ y AC-7, los algortitmos AC6-3+ y AC-7+ conservan las cuatro propiedades deseables. Mantienen además la nueva propiedad deseable, por las propiedades del Mínimo Soporte Inferior y Cero-Cardinalidad. Dado un mismo ordenamiento (de valores, variables y arcos), todo chequeo realizado por AC6-3+ es realizado por AC6++ y todo chequeo realizado por AC-7+ es realizado también por AC-7, sin embargo AC6-3+ y AC-7 evitan chequeos de valores deducidos como no viables, que no necesariamente son evitados por AC6++ y AC-7.

5. RESULTADOS EXPERIMENTALES

En esta sección se muestran algunos resultados de los experimentos realizados al comparar los rendimientos de AC6++ vs. AC6-3+ y AC-7 vs. AC-7+. En los experimentos se contó el número

de chequeos de restricciones realizados por cada algoritmo. Cada comparación se realizó con cincuenta instancias de cada problema.

En el problema ZEBRA [10] se obtuvo los siguientes resultados: AC6++: 717 chequeos y AC6-3+: 636 chequeos, AC-7: 640 chequeos y AC-7+: 594 chequeos. En el problema de la teoría combinatoria sugerido en [10], se obtuvo: AC6++ : 977 chequeos y AC6-3+: 783 chequeos, AC-7 : 966 chequeos y AC-7+: 826 chequeos.

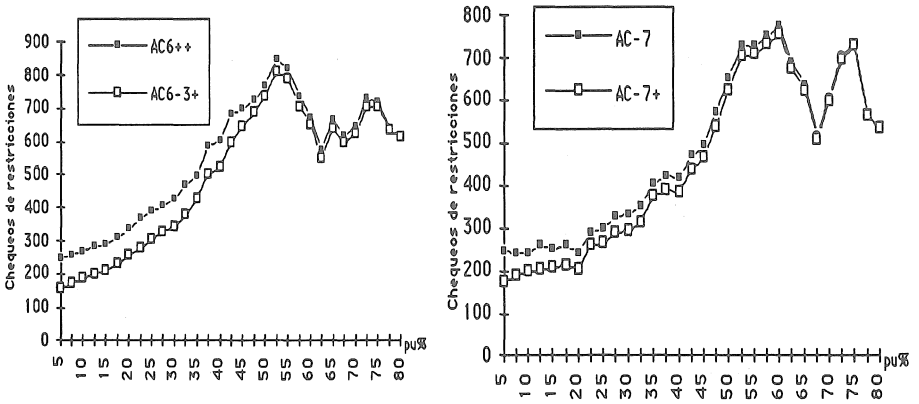

Figura 5. AC6++ vs. AC6-3+ (Izq.) y AC-7 vs. AC-7+ (Der.) en problemas aleatorios con n=20, d= 5, pc=30% y $0.05 \leq pu \leq 0.8$.

En los problemas aleatorios [1], n es el número de variables, d es el tamaño del dominio más grande, pc la probabilidad de que exista una restricción entre cualquier par de variables, y pu la probabilidad de que un par de valores cumplan una restricción. En la Figura 5, se muestran algunos resultados. En promedio, el rendimiento de AC6-3+ y AC-7+ es alrededor del 10% superior al de AC6++ y AC-7, respectivamente. Además, se observó que el número de valores deducidos como no viables fue proporcional al porcentaje de mejora. Aún cuando el número de valores deducidos como no viables fue pequeño, el número de chequeos fue apreciablemente reducido.

6. CONCLUSIONES

Se definió una nueva propiedad deseable para reducir el número de chequeos de restricciones realizados por un algoritmo de AC. Para considerar esta propiedad deseable, se definieron dos propiedades de deducción de no viabilidad. Como esta propiedad depende de la no viabilidad de los

valores, es muy práctica en problemas con fuertes propiedades estructurales (i.e., muchas restricciones, restricciones fuertes, muchas variables, dominios grandes, etc.). Estas propiedades son fácilmente adaptables a los dos mejores algoritmos de AC previamente conocidos, mejorando apreciablemente su comportamiento. Trabajos futuros en cálculo de Arco-Consistencia deben considerar esta nueva propiedad deseable.

REFERENCIAS

[1] Bessière, C. and Cordier, M. Arc Consistency and Arc Consistency again. In: Artificial Intelligence. Vol. 65 (1994) p. 179-190.

[2] Bessière, C. ; Frueder E.C. and Regin J.C. Using Inference to Reduce Arc Consistency Computation. In: *Proc.* ECAI'94.

[3] Bessière, C., Regin, J.C. Using bidirectionality to speed up arc-consistency processing. In: *Proc.* ECAI'94.

[4] Freuder E.C. Using metalevel knowledge to reduce constraint checking. In: *Proc.* ECAI'94.

[5] Mackworth, A.K. Consistency in networks of relations. In: Artificial Intelligence. Vol. 8. No. 1 (1977). p. 99-118.

[6] Mohr, R. and Henderson, T.C. Arc and path consistency revisited. In: Artificial Intelligence. Vol. 28 (1986). p. 225-223.

[7] Sabin, D. and Frueder E.C. Contradicting conventional wisdom in constraint satisfaction. In: *Proc.* ECAI'94.

[8] Valencia, F. and Castaño, G. Problemas de satisfacción de restricciones: Unificación y nuevos algoritmos. AV-95-03, Grupo AVISPA, Universidad Javeriana de Cali, 1995.

[9] Van Hentenryck, P. and Deville, Yves. An efficient Arc Consistency Algorithm for a Class of CSP Problems. p. 325-330. In: *Proc.* IJCAI'91.

[10] Van Hentenryck, P. Constraint Satisfaction in Logic Programing. MIT Press, 1989.

[11] Waltz, D. Generating semantic descriptions from drawing of scenes with shadows. Technical Report A1271, MIT, MA, 1972.