

**Algoritmo de consistencia de arcos**

Pablo Barrientos Lobato  
Adrián Infantes Moreiras  
Inteligencia Artificial I

Contenido

[Introducción 2](#_Toc410077896)

[Algoritmo AC-3 2](#_Toc410077897)

[**Globales.lisp** 2](#_Toc410077898)

[Variables globales 2](#_Toc410077899)

[Estructuras 3](#_Toc410077900)

[**AC3.lisp** 4](#_Toc410077901)

[**Random-2.lisp** 6](#_Toc410077902)

[**Random-n.lisp** 8](#_Toc410077903)

[**Mates.lisp** 9](#_Toc410077904)

[**Ejecutar-ac3-binarias.lisp** 9](#_Toc410077905)

[**Ejecutar-ac3-multiples.lisp** 10](#_Toc410077906)

[**Robot-ac3.lisp** 11](#_Toc410077907)

[Algoritmo AC-2001 12](#_Toc410077908)

[**AC2001.lisp** 12](#_Toc410077909)

[**GAC2001.lisp** 15](#_Toc410077910)

[**Ejecutar-AC2001-binarias.lisp** 19](#_Toc410077911)

[**Ejecutar-AC2001-multiples.lisp** 20](#_Toc410077912)

[Comparaciones y estadísticas 21](#_Toc410077913)

[**AC3-estadisticas.lisp** 21](#_Toc410077914)

[**AC2001-estadisticas.lisp** 22](#_Toc410077915)

[**Ejecutar-juntos-binarias//multiples.lisp** 23](#_Toc410077916)

[Conclusiones 25](#_Toc410077917)

[Bibliografía 29](#_Toc410077918)

[Documentos adjuntos 29](#_Toc410077919)

# Introducción

En este trabajo se exponen las implementaciones de los algoritmos AC3 y AC2001 de consistencia de arcos aplicados para problemas de satisfacción de restricciones, además de las versiones generalizadas de estos algoritmos para poder tratar problemas de cualquier aridad.

Por otra parte se muestra cómo se ha implementado un generador aleatorio de problemas de satisfacción de restricciones binarias y otro generalizado (para cualquier aridad) con el objetivo de realizar un análisis comparativo entre los algoritmos, teniendo en cuenta comprobaciones y tiempos medios de ejecución para una muestra significativa de problemas.

# Algoritmo AC-3

En este apartado pasaremos a explicar el código realizado para implementar el algoritmo AC-3 visto en clase, así como algunos ejemplos de su uso. Esta sección estará definida según los archivos que hemos enviado como documentos adjuntos.

## **Globales.lisp**

En el archivo encontramos las estructuras y variables globales que se usan en los problemas.

### Variables globales

**(**defvar \*variables\***)**

Esta variable es un mapa de dispersión en el que la clave es el nombre de la variable y el valor es la estructura de tipo variable.

**(**defvar \*variables-copia\***)**

Esta variable contiene lo mismo que la variable global anterior y se utiliza al ejecutar un mismo problema para los algoritmos ac-3 y ac-2001 (es auxiliar).

**(**defvar \*restricciones\***)**

Variable global que almacena todas las restricciones del PSR en una lista.

**(**defvar \*contador-arcos\***)**

Variable global que cuenta los arcos que se van procesando.

**(**defvar \*tiempo0\***)** y **(**defvar \*tiempo1\***)**

Variables que se usan para medir el tiempo de ejecución del problema

**(**defvar \*pares-prohibidos\***)**

Variable global que contiene una tabla de dispersión que almacena los pares prohibidos para cada restricción del problema. Esta tabla tiene como clave la lista de variables y como valor una lista de listas en el que cada elemento es el par de valores no admitidos correspondientes a las variables de la restricción.

**(**defvar \*tuplas-prohibidas\***)**

Es una tabla de dispersión global que se utilizará para guardar las tuplas de valores prohibidos para cada restricción del problema. La clave será la lista de variables y el valor asociado una lista de listas en la que cada elemento es la tupla de valores no permitidos correspondientes a las variables de la restricción.

**(**defvar \*ultimos-soportes\***)**

Tabla de dispersión usada en el AC-2001. Su clave está compuesta por la terna (vi, vj, valor-vi) y el valor es el soporte de vi en vj.

**(**defvar \*delete\***)**

Variable global que se utiliza para saber si se ha cambiado el dominio o no. Se usa en el AC-2001.

### Estructuras

Para este problema usaremos tres estructuras. Cada estructura se ha asociado con una función que se encarga de imprimir por consola la estructura de una forma que se pueda entender mejor. En esta memoria omitiremos esta función.

**(**defstruct **(**psr-var **(:print-function** escribe-psr-var**))**

nombre

dominio**)**

Esta estructura define una variable en el problema de satisfacción de restricciones. Consta de dos campos, uno es el nombre, y el otro es el dominio, que será del tipo array con los posibles valores de la variable en cuestión.

**(**defstruct **(**psr-restr **(:print-function** escribe-psr-restr**))**

variables

funcion**)**

Esta estructura definirá una restricción en el problema. Consta de dos campos. Uno es la lista de variables que componen la restricción, y el otro es el símbolo de la función que sirve para comprobar si la restricción se cumple o no.

**(**defstruct **(**arco **(:print-function** escribe-arco**))**

variable

restriccion**)**

Estructura que define un arco en el problema. Consta de dos campos:

* Variable : El nombre de la variable distinguida del arco
* Restricción: La restricción correspondiente al arco

## **AC3.lisp**

En este archivo están definidas las funciones arcos actualiza-dominio, ac-3, variables-arcos y valor-consistente.

La función arcos crea todos los arcos posibles a partir de la lista de restricciones del problema guardada en la variable global restricciones.

Pseudocódigo:

FUNCION ARCOS()

1. Hacer RESULTADO igual a vacío

2. Para cada RESTRICCION es \*RESTRICCIONES\*

2.1 Para cada VARIABLE en PSR-RESTR-VARIABLES(RESTRICCION)

Añadir a RESULTADO el arco formado por VARIABLE + RESTRICCION

3. Devolver RESULTADO

La función actualiza dominio es la que se encarga de actualizar el dominio de una variable dada una restricción. Esta función requiere de tres parámetros de entrada. Estos son:

· VARIABLE: El nombre de la variable de la que se quiere calcular su nuevo dominio.

· RESTRICCIÓN: La restricción sobre la cuál calcular su nuevo dominio.

· VARIABLES: La lista de variables del problema.

Pseudocódigo:

FUNCION ACTUALIZA-DOMINIO(VARIABLE,RESTRICCION,VARIABLES)

1. Hacer DOMINIO-ACTUAL igual al dominio de VARIABLE en VARIABLES

Hacer NUEVO-DOMINIO igual a vacío

2. Para cada VALOR en DOMINIO-ACTUAL hacer

2.1 Si para ese VALOR de VARIABLE existe al menos una asignación a las

restantes variables de RESTRICCION que satisfaga RESTRICCION,

incluir VALOR en NUEVO-DOMINIO

3. Devolver NUEVO-DOMINIO

La función AC-3 es que implementa el algoritmo en sí mismo. Tiene como parámetros de entrada las variables a tratar, guardadas en la variable global.

Devuelve las variables con los dominios actualizados. Hay que destacar que esta función sirve tanto para el algoritmo de restricciones binarias como para las restricciones múltiples.

Pseudocódigo:

FUNCION AC-3(VARIABLES)

1. Hacer RED igual a ARCOS()

2. Mientras RED no sea vacío

2.1 Hacer ACTUAL el primero de RED y RED el resto de RED

2.2 Hacer VARIABLE igual a ARCO-VARIABLE(ACTUAL) y

RESTRICCION igual ARCO-RESTRICCION(ACTUAL)

2.3 Hacer DOMINIO-ACTUAL igual al dominio

de VARIABLE en VARIABLES

2.4 Hacer DOMINIO-NUEVO igual a

ACTUALIZA-DOMINIO(VARIABLE,RESTRICCION,VARIABLES)

2.5 Si DOMINIO-NUEVO y DOMINIO-ACTUAL son distintos, entonces

2.5.1 Actualizar (destructivamente) el dominio de ACTUAL

en VARIABLES con NUEVO-DOMINIO

2.5.2 Incluir en RED todos los arcos de ARCOS() en los

que aparezca VARIABLE y ésta NO sea la variable

distinguida del arco (\*)

3. Devolver VARIABLES

La siguiente función construye una tabla de dispersión donde aparecen como claves las variables del PSR y como valores la lista de arcos de esa variable en las que aparece como variable no distinguida. La función tiene como parámetros:

· ARCOS: La red de arcos original como una lista

· VARIABLES-ORIGINAL: La tabla de dispersión con todas las variables originales.

Se devuelve una tabla de dispersión con los pares (variable, lista de arcos a los que afecta).

Pseudocódigo:

FUNCION VARIABLES-ARCOS(ARCOS VARIABLES-ORIGINALES)

1. Hacer RESULTADO igual a MAPHASH()

Hacer VARIABLES igual a la lista de las claves de VARIABLES-ORIGINALES

2. Para cada VARIABLE en VARIABLES hacer

2.1 Para cada ARCO en ARCOS hacer

2.1.1 Hacer VARIABLE-DISTINGUIDA igual a (ARCO-VARIABLE ARCO)

Hacer VARIABLES-TODAS igual a las variables de la restricción del ARCO

HACER VALORES igual a la lista de arcos que ya tiene asignados VARIABLE

2.1.2 SI NO es igual VARIABLE que VARIABLE-DISTINGUIDA AND VARIABLE pertenece a VARIABLES todas

2.1.2.1 Añadir a VALORES el arco ARCO

3 Devolver RESULTADO

La siguiente función es la que genera todas las tuplas posibles de una serie de valores asignados a variables y comprueba que dichos valores cumplen cierta restricción. Si existe una tupla que cumple esto último el algoritmo acaba devolviendo T. Para ello se van a explorar todos los posibles valores de todos los dominios excepto de una variable especial a la que sólo le asignamos un valor pasado como argumento.

Los parámetros son:

· VALOR: El valor de la variable especial

· VARIABLE: La variable especial

· RESTRICCIÓN: La restricción que hay que comprobar.

· VARIABLES: Variables originales con sus respectivos dominios.

· OTRAS-VARIABLES: Lista de variables que aún quedan por asignarles un valor

· TABLA: Tabla de dispersión con las asociaciones realizadas (variable, valor)

Devuelve T si encuentra una asignación consistente con RESTRICCIÓN, o nil en caso contrario.

Pseudocódigo:

FUNCION VALOR-CONSISTENTE(VALOR VARIABLE RESTRICCION VARIABLES OTRAS-VARIABLES TABLA)

1. SI OTRAS-VARIABLES está vacío

1.1 Devolver la comprobación de que RESTRICCIÓN se cumple o no.

2. SINO

2.1 HACER RESULTADO igual a NIL

VARIABLE-ACTUAL la primera variable de OTRAS-VARIABLES

2.2 SI es igual VARIABLE que VARIABLE-ACTUAL

2.2.1 Añadir a TABLA el par (VARIABLE,VALOR)

2.2.2 HACER RESULTADO igual a VALOR-CONSISTENTE(VALOR VARIABLE RESTRICCION VARIABLES (RESTO DE OTRAS-VARIABLES) TABLA)

2.3 SINO

2.3.1 Para cada VALOR-ACTUAL en el dominio de VARIABLE-ACTUAL hacer

2.3.1.1 Añadir a TABLA el par (VARIABLE-ACTUAL,VALOR-ACTUAL)

2.3.1.2 HACER RESULTADO igual a VALOR-CONSISTENTE(VALOR VARIABLE RESTRICCION VARIABLES (RESTO DE OTRAS-VARIABLES) TABLA)

2.3.1.3 SI RESULTADO es T Devolver RESULTADO

2.3.2 Devolver RESULTADO

## **Random-2.lisp**

En esta clase se incluyen los métodos necesarios para generar un problema PSR binario de forma aleatoria. Se compone de cuatro métodos.

La función “comprueba-pares-prohibidos” es una función que, dada una asignación de valores en el parámetro de entrada TABLA, comprueba si esa asignación es una asignación válida para el problema o por el contrario se encuentra entre los pares prohibidos.

Pseudocódigo:

FUNCION COMPRUEBA-PARES-PROHIBIDOS(TABLA)

1. Hacer VARIABLES igual a la lista ordenada de todas las variables de la asignación

Hacer VALORES igual a la lista de todos los valores asignados a esas variables

2. Devolver la comprobación de que el par (VARIABLES,VALORES) NO está entre los pares prohibidos de esas variables

La siguiente función tiene como objetivo crear todas las restricciones posibles según el número de variables del problema. Esta función será usada luego en el método que genera el problema de PSR para generar todas las posibles restricciones e ir eliminando aleatoriamente hasta que se quede solo el número de restricciones deseadas.

Pseudocódigo:

FUNCION CREA-TODAS-RESTRICCIONES(NVARIABLES)

1. Hacer RESULTADO igual a la lista vacía

2. Para cada I desde 1 hasta NVARIABLES hacer

2.1. Para cada J desde I+1 hasta NVARIABLE hacer

2.1.1. Añadir a RESULTADO la restricción creada con la lista de variables (I J)

3. Hacer \*RESTRICCIONES\* igual a RESULTADO

La siguiente función genera todas las tuplas posibles de aridad n, cada una de ellas con un rango de números naturales que va desde 1 hasta el dominio. Esta función también es usada en el método que genera el problema aleatorio.

Pseudocódigo:

FUNCION GENERA-PSR-N(NVARIABLES NDOMINIO NRESTRICCIONES NPARES-PROHIBIDOS NARIDAD)

1. SI NARIDAD = 0

1.1 Añadir TUPLA a LISTA

1.2 Devolver LISTA

2. SINO

2.1 Desde I = 1 hasta NDOMINIO

2.2 Hacer LISTA igual a GENERA-TODAS-TUPLAS(NDOMINIO NARIDAD-1 LISTA (CONS I TUPLA))

2.3 Devolver LISTA

La siguiente función es la encargada de generar un problema PSR aleatorio con restricciones binarias. Este método tiene los siguientes parámetros:

* NVARIABLES es el número de variables del PSR
* NDOMINIO es el tamaño del dominio de todas las variables
* NRESTRICCIONES es el número de restricciones del problema
* NPARES-PROHIBIDOS es el número de pares prohibidos de valores para todas la restricciones

Hay que aclarar que este método da error si se sobrepasa el número máximo de restricciones que puede haber en el problema o el máximo número posible de pares prohibidos.

FUNCION GENERA-PSR-2(NVARIABLES NDOMINIO NRESTRICCIONES NPARES-PROHIBIDOS)

1. Hacer \*VARIABLES\* igual a la tabla de dispersión vacía

2. Hacer \*PARES-PROHIBIDOS\* igual a la tabla de dispersión vacía

3. Hacer DOMINIO igual a una lista con los valores 1 ... NDOMINIO

4. Para cada VARIABLE desde 1 hasta NVARIABLES hacer

4.1. Crear variable con nombre VARIABLE y dominio DOMINIO

5. Hacer \*RESTRICCIONES\* igual a (CREAR-TODAS-RESTRICCIONES)

6. Mientras TAMAÑO(\*RESTRICCIONES\*) > NRESTRICCIONES hacer

6.1. Eliminar aleatoriamente una restricción de \*RESTRICCIONES\*

7. Para cada RESTRICCION en \*RESTRICCIONES\* hacer

7.1. Hacer LISTA-PARES-PROHIBIDOS igual a la lista con todos los posibles pares

7.2. Mientras TAMAÑO(LISTA-PARES-PROHIBIDOS) > NPARES-PROHIBIDOS hacer

7.2.1. Eliminar aleatoriamente un par de LISTA-PARES-PROHIBIDOS

7.3. Crear par (RESTRICCION,LISTA-PARES-PROHIBIDOS) en \*PARES-PROHIBIDOS\*

## **Random-n.lisp**

En este archivo se encuentran los métodos para generar un PSR aleatorio de aridad n.

La función “comprueba-tuplas-prohibidas” hace el mismo trabajo que la función de “comprueba-pares-prohibidos” en el archivo de random-2.lisp, que es comprobar si la asignación parcial del problema es prohibida o no.

El pseudocódigo es el siguiente:

FUNCION COMPRUEBA-TUPLAS-PROHIBIDAS(TABLA)

1. Hacer VARIABLES igual a la lista ordenada de todas las variables de la asignación

Hacer VALORES igual a la lista de todos los valores asignados a esas variables

2. Devolver la comprobación de que el par (VARIABLES,VALORES) NO está entre los pares prohibidos de esas variables

La siguiente función es la encargada de crear todas las restricciones posibles para un problema de aridad n. Hemos de destacar que, para la creación de este método, hemos usado tres métodos auxiliares que hemos encontrado en internet, y que se encargan de crear todas las combinaciones posibles (funciones “n-combination”, “index-append-combination” y “combination”). La dirección de la que lo hemos sacado se encuentra en la bibliografía.

El pseudocódigo es el siguiente:

FUNCION CREA-TODAS-RESTRICCIONES(NVARIABLES NARIDAD)

1. Hacer RESULTADO igual a la lista vacía

2. Para cada LISTA-VARIABLES de todas las posibles combinaciones

2.1. Añadir a RESULTADO la restricción creada con la lista de variables LISTA-VARIABLES

3. Hacer \*RESTRICCIONES\* igual a RESULTADO

La siguiente función es la encargada de generar el PSR de aridad n aleatorio. Tiene los siguientes parámetros:

* NVARIABLES: El número de variables del PSR
* NDOMINIO: El tamaño del dominio de todas las variables
* NRESTRICCIONES: El número de restricciones del problema
* NTUPLAS-PROHIBIDAS: El número de pares prohibidos de valores para todas la restricciones

Al igual que el método encargado de generar un PSR aleatorio binario, esta función se encarga de comprobar si existen errores en los parámetros de entrada de restricciones, de tuplas prohibidas y, además, comprueba que la aridad no sea mayor a las variables del PSR.

El pseudocódigo es el siguiente:

FUNCION GENERA-PSR-N(NVARIABLES NDOMINIO NRESTRICCIONES NPARES-PROHIBIDOS NARIDAD)

1. Hacer \*VARIABLES\* igual a la tabla de dispersión vacía

2. Hacer \*TUPLAS-PROHIBIDAS\* igual a la tabla de dispersión vacía

3. Hacer DOMINIO igual a una lista con los valores 1 ... NDOMINIO

4. Para cada VARIABLE desde 1 hasta NVARIABLES hacer

4.1. Crear variable con nombre VARIABLE y dominio DOMINIO

5. Hacer \*RESTRICCIONES\* igual a (CREAR-TODAS-RESTRICCIONES)

6. Mientras TAMAÑO(\*RESTRICCIONES\*) > NRESTRICCIONES hacer

6.1. Eliminar aleatoriamente una restricción de \*RESTRICCIONES\*

7. Para cada RESTRICCION en \*RESTRICCIONES\* hacer

7.1. Hacer LISTA-TUPLAS-PROHIBIDAS igual a la lista con todas las posibles tuplas

7.2. Mientras TAMAÑO(LISTA-TUPLAS-PROHIBIDAS) > NTUPLAS-PROHIBIDAS hacer

7.2.2. eliminar aleatoriamente un elemento de la lista LISTA-TUPLAS-PROHIBIDAS

7.3. Crear par (RESTRICCION,LISTA-TUPLAS-PROHIBIDAS) en \*TUPLAS-PROHIBIDAS\*

La siguiente función es la encargada de generar todas las tuplas de aridad n posibles, cada una de ellas con un rango de números naturales que va desde 1 hasta el dominio.

El pseudocódigo es el siguiente:

FUNCION GENERA-TODAS-TUPLAS(NDOMINIO NARIDAD LISTA TUPLA)

1. SI NARIDAD = 0

1.1 Añadir TUPLA a LISTA

1.2 Devolver LISTA

2. SINO

2.1 Desde I = 1 hasta NDOMINIO

2.2 Hacer LISTA igual a GENERA-TODAS-TUPLAS(NDOMINIO NARIDAD-1 LISTA (CONS I TUPLA))

2.3 Devolver LISTA

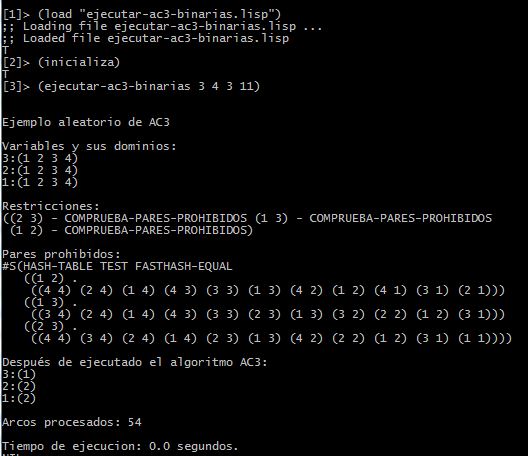
## **Mates.lisp**

Este archivo contiene funciones matemáticas auxiliares. Las funciones sirven para calcular el factorial de un número, para calcular el número de combinaciones sin repetición de N elementos tomados de M en M, y una función que calcula el número de combinaciones con repetición de N elementos tomados de M en M. Estas dos últimas funciones se usan para ver el número máximo posible de restricciones y de tuplas prohibidas que existen en un PSR dependiendo del número de variables.

## **Ejecutar-ac3-binarias.lisp**

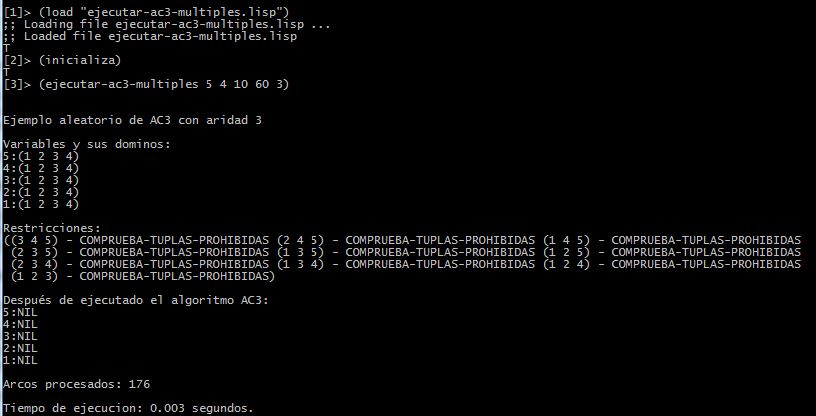
En este archivo se crea la función que servirá para ejecutar el algoritmo de ac-3 en la consola de lisp. El archivo contiene la función inicializa, que se encarga de cargar los archivos anteriormente mencionados, y otra función que genera un PSR binario, lo formatea por consola de forma que podamos ver el problema generado, lo resuelve y nos muestre por pantalla el resultado del algoritmo, así como los arcos que ha procesado y el tiempo de ejecución del mismo.

En la siguiente imagen podemos ver un ejemplo:



## **Ejecutar-ac3-multiples.lisp**

Este archivo es análogo al anterior, solo que genera un PSR de aridad n. Podemos ver un ejemplo tanto de su uso como del resultado en la siguiente imagen:

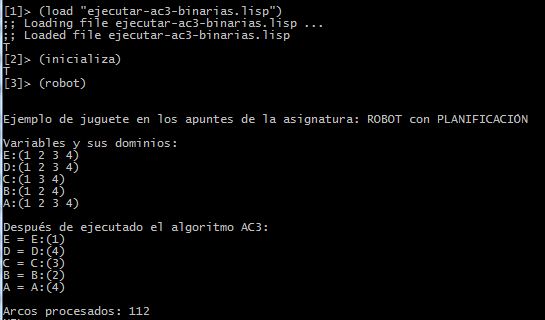


## **Robot-ac3.lisp**

Este archivo contiene el problema del robot visto en clase, para comprobar que el algoritmo AC-3 funciona de forma correcta.

Para ello, hemos creado una función por cada restricción existente en el problema y, en la función robot, hemos inicializado la variable global variables con las variables iniciales del problema, así como las restricciones.

Una vez tenemos inicializadas las variables globales, llamamos a la función ac-3 y mostramos el resultado por pantalla, comprobando que el problema da el resultado correcto:



# Algoritmo AC-2001

A continuación se detalla la implementación del algoritmo AC-2001 organizado en los distintos archivos necesarios para su correcto funcionamiento. Para ello nos hemos basado en el documento anexo en la bibliografía *“An optimal coarse-grained arc consistency algorithm”* que relata información muy relevante acerca del algoritmo, así como datos y pruebas realizadas.

## **AC2001.lisp**

Función que crea todos los arcos posibles de un PSR a partir de la lista de restricciones.

FUNCION ARCOS2001()

1. Hacer RESULTADO igual a vacío

2. Para cada RESTRICCION es \*RESTRICCIONES\*

2.1 Para cada VARIABLE en PSR-RESTR-VARIABLES(RESTRICCION)

Añadir a RESULTADO el arco formado por VARIABLE + RESTRICCION

3. Devolver RESULTADO

Función que, dada una variable y una restricción binaria, devuelve la otra variable que participa en la restricción. Esta función tiene los siguientes parámetros:

· VARIABLE: Una de las variables de la RESTRICCION.

· RESTRICCION: La restricción sobre las que estamos operando

La función devuelve la otra variable de la restricción que se le pasa por parámetro.

Pseudocódigo:

FUNCION GET-OTRA-VARIABLE(VARIABLE RESTRICCION)

1. Hacer v1 igual a la primera variable de PSR-RESTR-VARIABLES(RESTRICCION)

Hacer v2 igual a la segunda variable de PSR-RESTR-VARIABLES(RESTRICCION)

2. Si VARIABLE = v1

2.1 Devolver v2

Si no

2.2 Devolver v1

La función succ devuelve el siguiente índice del array ordenado con los valores del dominio de una variable que tenga un elemento distinto de NIL a partir del índice i (sin incluir). La función tiene los siguientes parámetros:

· I: Índice de la tabla a partir del cual queremos calcular el sucesor.

· DOMINIO: Lista ordenada ascendentemente.

En el caso de que el índice no exista la función devuelve NIL

Pseudocódigo:

FUNCION SUCC(I DOMINIO)

1. Hacer FIN igual a tamaño del array dominio - 1

Hacer COMIENZO igual a v si v es ditinto de NIL, 0 en otro caso.

Hacer RESULTADOS igual a NIL

2. Desde I = COMIENZO hasta FIN

2.1 Si DOMINIO[I] es distinto de NIL

Devolver I

3. Devolver NIL

Función que actualiza el dominio de una variable dependiendo de una restricción (versión AC2001). Esta función toma como parámetros:

· VARIABLE: El nombre de la variable de la que se quiere calcular su nuevo dominio.

· RESTRICCION: La restricción sobre la cual se va a calcular el nuevo dominio.

· VARIABLES: Lista de variables del PSR.

La función devuelve la lista con los valores del nuevo dominio de la variable

Pseudocódigo:

FUNCION ACTUALIZA-DOMINIO(VARIABLE,RESTRICCION,VARIABLES)

1. Hacer DOMINIO-ACTUAL igual al dominio de VARIABLE en VARIABLES

Hacer NUEVO-DOMINIO igual al dominio de VARIABLE en VARIABLES

Hacer OTRA-VARIABLE igual a GET-OTRA-VARIABLE(VARIABLE, RESTRICCION)

2. Para cada valor A en DOMINIO-ACTUAL hacer

2.1 Hacer B el soporte de OTRA-VARIABLE en (VARIABLE, A)

Hacer DOMINIO-OTRA-VARIABLE igual al dominio de OTRA-VARIABLE en VARIABLES

2.2 Si B no está en DOMINIO-OTRA-VARIABLE

2.2.1 Hacer B igual al SUCC(B, DOMINIO-OTRA-VARIABLE)

2.2.2 MIENTRAS B distinto de NIL Y NO VALOR-CONSISTENTE2001(VARIABLE, OTRA-VARIABLE A B RESTRICCION)

2.2.2.1 Hacer B igual al SUCC(B, DOMINIO-OTRA-VARIABLE)

2.2.3 SI B distinto de NIL

2.2.3.1 Hacer B último soporte de OTRA-VARIABLE en (VARIABLE, A)

SI NO

2.2.3.2 Quitar A de NUEVO-DOMINIO

3. Devolver NUEVO-DOMINIO

Función que calcula si una pareja de asignaciones (variable, valor) es un par de valores permitidos para una restricción binaria. Los parámetros de esta función son:

· VI: El nombre de la primera variable.

· VJ: El nombre de la segunda variable.

· A: El valor que tiene la variable VI en la asignación.

· B: El valor que tiene la variable VJ en la asignación.

· RESTRICCION: La restricción cuyo par de valores hay que comprobar.

La función devuelve T en el caso de que la asignación sea consistente, nil en caso contrario.

Pseudocódigo:

FUNCION VALOR-CONSISTENTE2001(VI VJ A B RESTRICCION)

1. Hacer TABLA la tabla de dispersión vacía

2. Añadir a TABLA el par (vi,a)

3. Añadir a TABLA el par (vj,b)

4. Devolver PSR-RESTR-FUNCION(RESTRICCION, TABLA)

Función que ejecuta el algoritmo AC-2001. Tiene como parámetros la variable global variables. Devuelve esa misma variable global actualizada con el resultado del problema.

FUNCION AC-2001(VARIABLES)

1. Hacer RED igual a ARCOS()

2. Mientras RED no sea vacío

2.1 Hacer \*ULTIMOS-SOPORTES\* igual la tabla de dispersión vacía

2.2 Hacer ACTUAL el primero de RED y RED el resto de RED

2.3 Hacer VARIABLE igual a ARCO-VARIABLE(ACTUAL) y

RESTRICCION igual ARCO-RESTRICCION(ACTUAL)

2.4 Hacer DOMINIO-ACTUAL igual al dominio

de VARIABLE en VARIABLES

2.5 Hacer DOMINIO-NUEVO igual a

ACTUALIZA-DOMINIO(VARIABLE,RESTRICCION,VARIABLES)

2.6 Si DOMINIO-NUEVO y DOMINIO-ACTUAL son distintos, entonces

2.6.1 Actualizar (destructivamente) el dominio de ACTUAL

en VARIABLES con NUEVO-DOMINIO

2.6.2 Incluir en RED todos los arcos de ARCOS() en los

que aparezca VARIABLE y ésta NO sea la variable

distinguida del arco (\*)

3. Devolver VARIABLES

Función que construye una tabla de dispersión donde aparecen como claves las variables del PSR y como valores la lista de arcos de esa variable en las que aparece como variable NO distinguida. La función toma como parámetros los siguientes:

· ARCOS: La red de arcos original como lista.

· VARIABLES-ORIGINALES: La tabla de dispersión con todas las variables originales.

Esta función devuelve una tabla de dispersión con los pares:

(variables, lista de arcos a los que afecta)

Pseudocódigo:

FUNCION VARIABLES-ARCOS2001(ARCOS VARIABLES-ORIGINALES)

1. Hacer RESULTADO igual a MAPHASH()

Hacer VARIABLES igual a la lista de las claves de VARIABLES-ORIGINALES

2. Para cada VARIABLE en VARIABLES hacer

2.1 Para cada ARCO en ARCOS hacer

2.1.1 Hacer VARIABLE-DISTINGUIDA igual a (ARCO-VARIABLE ARCO)

Hacer VARIABLES-TODAS igual a las variables de la restricción del ARCO

HACER VALORES igual a la lista de arcos que tiene asignados VARIABLE

2.1.2 SI NO es igual VARIABLE que VARIABLE-DISTINGUIDA AND VARIABLE pertenece a VARIABLES todas

2.1.2.1 Añadir a VALORES el arco ARCO

3 Devolver RESULTADO

## **GAC2001.lisp**

Función que crea todos los arcos posibles de un PSR a partir de la lista de restricciones almacenada en una variable global y la devuelve.

Pseudocódigo:

FUNCION ARCOS2001()

1. Hacer RESULTADO igual a vacío

2. Para cada RESTRICCION es \*RESTRICCIONES\*

2.1 Para cada VARIABLE en PSR-RESTR-VARIABLES(RESTRICCION)

Añadir a RESULTADO el arco formado por VARIABLE + RESTRICCION

3. Devolver RESULTADO

Función que devuelve, dada una tupla de valores, la siguiente tupla en orden lexicográfico inverso. Por ejemplo, si la tupla de partida es (1 2 3) la siguiente tupla sería (2 2 3). Si establecemos el máximo para cada valor en 4, las siguientes tuplas a partir de esta última serían (3 2 3) → (4 2 3) → (1 3 3) → ...

Esta función se compone de tres parámetros:

· MAXIMO: Máximo valor para cada elemento en la tupla.

· TUPLA: Tupla original.

· N: Número de elementos de la tupla a devolver.

Esta función debe devolver NIL en el caso de que hayamos llegado a la última tupla posible, o la tupla (1 1 … 1) en el caso de que la tupla que le pasemos por parámetro sea nula.

Pseudocódigo:

FUNCION SIGUIENTE-TUPLA(MAXIMO TUPLA N)

1. Si TUPLA es distinto de NIL

1.1 Si TUPLA = (MAXIMO MAXIMO ... MAXIMO)

1.1.1 Devolver NIL

1.2 Si no

1.2.1 Hacer PRIMER-ELEMENTO igual al primer elemento de TUPLA

1.2.2 Si PRIMER-ELEMENTO < MAXIMO

1.2.2.1 Devolver (CONS PRIMER-ELEMENTO+1 (CDR TUPLA))

1.2.3 Si no

1.2.3.1 Si N = 0

1.2.3.1.1 Devolver NIL

1.2.3.2 Si no

1.2.3.2.1 Devolver (CONS 1 SIGUIENTE-TUPLA(MAXIMO, (CDR TUPLA) (1- N)))

2. Si no

2.1 Devolver (1 1 ... 1)

Función que devuelve, dada una tupla, la tupla sucesora en orden lexicográfico. Sólo se permiten aquellas tuplas cuyos correspondientes valores sean válidos en los dominios de las variables implicadas en una restricción dada. Además, una determinada variable debe tener el valor pasado como argumento. Los parámetros de esta función son los siguientes:

· TUPLA: Tupla original.

· VARIABLES: El conjunto de variables.

· VARIABLE-ACTUAL: La variable a la que se le impone un valor.

· VALOR-ACTUAL: El valor que debe tomar la variable actual.

· RESTRICCION: La restricción que se está comprobando.

En el caso de que no exista una tupla sucesora que cumpla las restricciones del PSR, este método devuelve NIL.

Pseudocódigo:

FUNCION SUCC(TUPLA, VARIABLES, VARIABLE-ACTUAL, VALOR-ACTUAL, RESTRICCION)

1. Hacer DOMINIO igual al dominio de la VARIABLE-ACTUAL

Hacer NDOMINIO el número máximo de valores de DOMINIO

Hacer NARIDAD la aridad de la restricción

Hacer POSICION-VARIABLE-ACTUAL igual a la posición de la VARIABLE-ACTUAL en la lista de variables de la restricción

Hacer NUEVA-TUPLA igual a SIGUIENTE-TUPLA(NDOMINIO, TUPLA, NARIDAD)

Hacer FIN igual a NIL

2. Mientras NUEVA-TUPLA distinta de NIL y NO FIN

2.1 Si COMPRUEBA-DOMINIO(NUEVA-TUPLA,VARIABLES,RESTRICCION) Y

VALOR-ACTUAL = Valor del dominio de VARIABLE-ACTUAL apuntado por NUEVA-TUPLA

Hacer FIN igual a T

2.2 Si no

Hacer NUEVA-TUPLA igual a SIGUIENTE-TUPLA(NDOMINIO, TUPLA, NARIDAD)

3. Devolver NUEVA-TUPLA

Función que comprueba, dada una tupla, si todos los valores apuntados por los elementos de esa tupla pertenecen al dominio de cada una de las correspondientes variables que participan en cierta restricción. La función aceptará por parámetros:

· TUPLA: Tupla original.

· VARIABLES: El conjunto de variables.

· RESTRICCION: La restricción que se está comprobando.

Esta función devuelve T en el caso de que el dominio sea correcto o nil en otro caso.

Pseudocódigo:

FUNCION COMPRUEBA-DOMINIO(TUPLA VARIABLES RESTRICCION)

1. Hacer LISTA-VARIABLES igual a las variables de la RESTRICCION

Hacer RESULTADO igual a T

Hacer INDICE-EN-TUPLA igual a 0

2. Si TUPLA es distinta de NIL

2.1 Para cada VARIABLE en LISTA-VARIABLES hacer

2.1.1 Hacer DOMINIO igual al dominio de VARIABLE

Hacer INDICE-EN-DOMINIO igual al elemento INDICE-EN-TUPLA de TUPLA - 1

Hacer RESULTADO igual al valor del DOMINIO en INDICE-DOMINIO

2.1.2 Si RESULTADO es NIL

Devolver RESULTADO

2.1.3 Incrementar INDICE-EN-TUPLA

3. Devolver RESULTADO

Función que comprueba, dada una tupla, si todos los valores apuntados por los índice de esa tupla están o no prohibidos para una restricción dada. Los parámetros de esta función serán:

· TUPLA: Tupla original.

· VARIABLES: El conjunto de variables.

· RESTRICCION: La restricción que se está comprobando.

Esta función devuelve T en el caso de que la tupla sea consistente con la restricción dada o nil en caso contrario.

Pseudocódigo:

FUNCION VALOR-CONSISTENTE2001(TUPLA VARIABLES RESTRICCION)

1. Hacer TABLA igual a la tabla de dispersión vacía

Hacer INDICE-EN-TUPLA igual a 0

2. Para cada VARIABLE en las variables de RESTRICCION hacer

2.1 Hacer DOMINIO igual al dominio de VARIABLE

Hacer INDICE-EN-DOMINIO igual al elemento INDICE-EN-TUPLA de TUPLA - 1

Hacer VALOR igual al valor del DOMINIO en INDICE-DOMINIO

2.2 Introducir en TABLA la asociación (VARIABLE,VALOR)

2.3 Incrementar INDICE-EN-TUPLA

3. Devolver la llamada a la función descrita en la RESTRICCION con TABLA

Función que devuelve el dominio actualizado de una variable dependiendo de una restricción (versión GAC2001). Esta función tiene los siguientes parámetros:

· VARIABLE: El nombre de la variable de la que se quiere calcular su nuevo dominio.

· RESTRICCION: La restricción sobre la cual calcular el nuevo dominio.

· VARIABLES: Lista de variables del PSR.

Pseudocódigo:

FUNCION ACTUALIZA-DOMINIO(VARIABLE,RESTRICCION,VARIABLES)

1. Hacer DOMINIO-ACTUAL igual al dominio de VARIABLE en VARIABLES

Hacer NUEVO-DOMINIO igual al dominio de VARIABLE en VARIABLES

Hacer LIMITE igual al tamaño del DOMINIO-ACTUAL - 1

2. Desde I = 0 hasta LIMITE hacer

2.1 Hacer A el valor I del DOMINIO-ACTUAL

Hacer TUPLA-SOPORTE el valor anterior de la tupla soporte para la clave (VARIABLE, RESTRICCION, A)

2.2 Si A distinto de NIL Y (TUPLA-SOPORTE distinta de NIL O NO COMPRUEBA-DOMINIO(TUPLA-SOPORTE,VARIABLES,RESTRICCION))

2.2.1 Hacer TUPLA-SOPORTE igual al SUCC(TUPLA-SOPORTE, VARIABLES, VARIABLE, A, RESTRICCION)

2.2.2 MIENTRAS TUPLA-SOPORTE distinto de NIL Y NO VALOR-CONSISTENTE2001(TUPLA-SOPORTE, VARIABLES, RESTRICCION)

2.2.2.1 Hacer TUPLA-SOPORTE igual al SUCC(TUPLA-SOPORTE, VARIABLES, VARIABLE, A, RESTRICCION)

2.2.3 SI TUPLA-SOPORTE distinto de NIL

2.2.3.1 Hacer TUPLA-SOPORTE último soporte de la clave (VARIABLE, RESTRICCION, A)

SI NO

2.2.3.2 Quitar A de NUEVO-DOMINIO

3. Devolver NUEVO-DOMINIO

Función AC-2001, correspondiente al algoritmo AC-2001. Esta función toma como parámetros las variables del PSR.

FUNCION AC-2001(VARIABLES)

1. Hacer RED igual a ARCOS()

2. Mientras RED no sea vacío

2.1 Hacer \*ULTIMOS-SOPORTES\* igual la tabla de dispersión vacía

2.2 Hacer ACTUAL el primero de RED y RED el resto de RED

2.3 Hacer VARIABLE igual a ARCO-VARIABLE(ACTUAL) y

RESTRICCION igual ARCO-RESTRICCION(ACTUAL)

2.4 Hacer DOMINIO-ACTUAL igual al dominio

de VARIABLE en VARIABLES

2.5 Hacer DOMINIO-NUEVO igual a

ACTUALIZA-DOMINIO(VARIABLE,RESTRICCION,VARIABLES)

2.6 Si DOMINIO-NUEVO y DOMINIO-ACTUAL son distintos, entonces

2.6.1 Actualizar (destructivamente) el dominio de ACTUAL

en VARIABLES con NUEVO-DOMINIO

2.6.2 Incluir en RED todos los arcos de ARCOS() en los

que aparezca VARIABLE y ésta NO sea la variable

distinguida del arco (\*)

3. Devolver VARIABLES

Función que construye y devuelve una tabla de dispersión donde aparecen como claves las variables del PSR y como valores la lista de arcos de esa variable en las que aparece como variable NO distinguida. Esta función toma los siguientes parámetros de entrada:

· ARCOS: La red de arcos original como lista.

·VARIABLES-ORIGINALES: La tabla de dispersión con todas las variables originales

Pseudocódigo:

FUNCION VARIABLES-ARCOS2001(ARCOS VARIABLES-ORIGINALES)

1. Hacer RESULTADO igual a MAPHASH()

Hacer VARIABLES igual a la lista de las claves de VARIABLES-ORIGINALES

2. Para cada VARIABLE en VARIABLES hacer

2.1 Para cada ARCO en ARCOS hacer

2.1.1 Hacer VARIABLE-DISTINGUIDA igual a (ARCO-VARIABLE ARCO)

Hacer VARIABLES-TODAS igual a las variables de la restricción del ARCO

HACER VALORES igual a la lista de arcos que ya tiene asignados VARIABLE

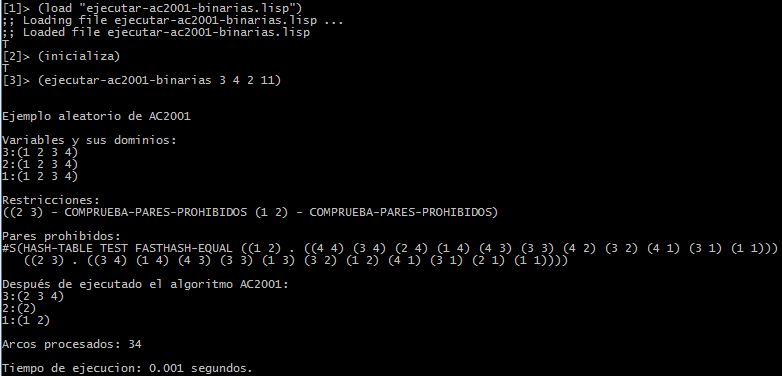
2.1.2 SI NO es igual VARIABLE que VARIABLE-DISTINGUIDA AND VARIABLE pertenece a VARIABLES todas

2.1.2.1 Añadir a VALORES el arco ARCO

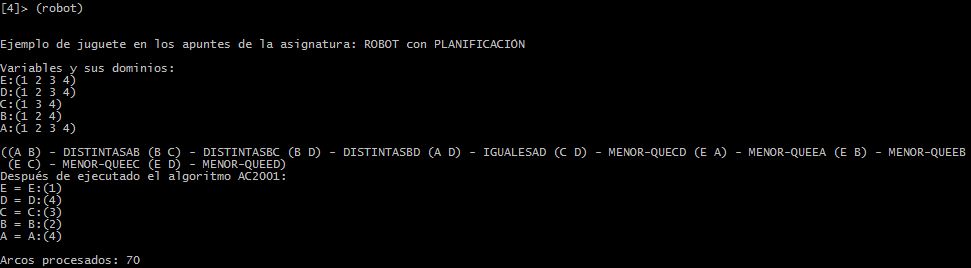
3 Devolver RESULTADO

## **Ejecutar-AC2001-binarias.lisp**

Este archivo es el primer archivo de ejecución del algoritmo AC2001. Después de cargar el archivo .lisp "ejecutar-ac2001-binarias.lisp" y ejecutar en comando inicializa() para los archivos complementarios necesarios, ejecutaremos la sentencia tipo:



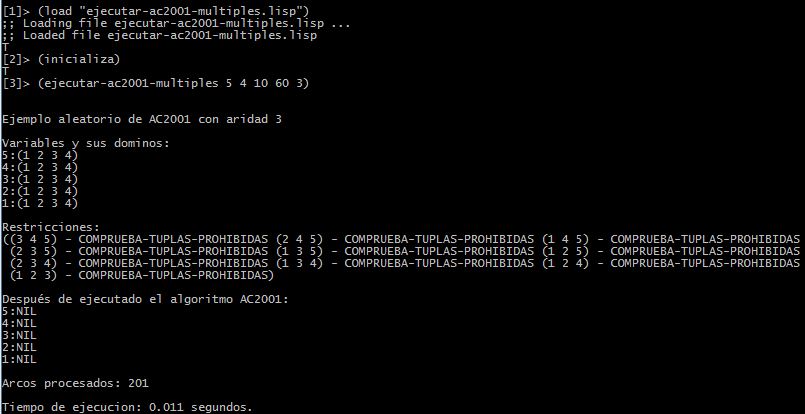
\*Ejemplo de ejecución del algoritmo ac2001 para restricciones binarias, de 3 variables, tamaño de dominio 4 , 2 restricciones y 11 tuplas prohibidas.



\*Ejemplo análogo al anterior pero ejecutado sobre el ejemplo específico del robot.

## **Ejecutar-AC2001-multiples.lisp**

Para ejecutar la versión generalizada del AC2001, cargamos el archivo .lisp "ejecutar-ac2001-multiples.lisp" , ejecutar inicializa() y una sentencia tipo:

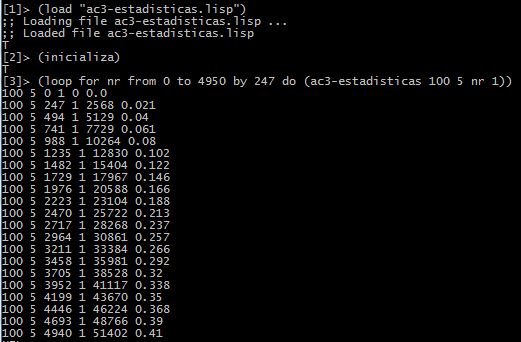


Donde tenemos los mismos parámetros que anteriormente excepto que se añade un último parámetro que es la aridad de las restricciones.

# Comparaciones y estadísticas

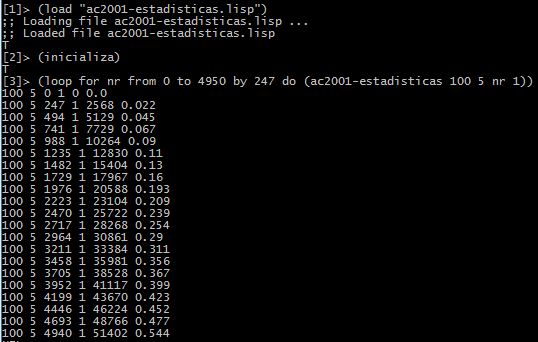
## **AC3-estadisticas.lisp**

Este archivo permite ejecutar una batería de problemas binarios aleatorios de AC-3 con el objetivo de poder crear gráficos. El archivo se compone de una función inicializa (carga los archivos de lisp compilados necesarios para crear y ejecutar un problema de AC3) y de una función que hace lo mismo que “ejecuta-ac3-binarias” sólo que formatea el resultado en forma de tabla para poder exportar los datos al Excel, tal y como se ve en la siguiente imagen:



## **AC2001-estadisticas.lisp**

Para ejecutar una batería de problemas de AC2001 aleatorios ejecutaríamos las siguientes sentencias:

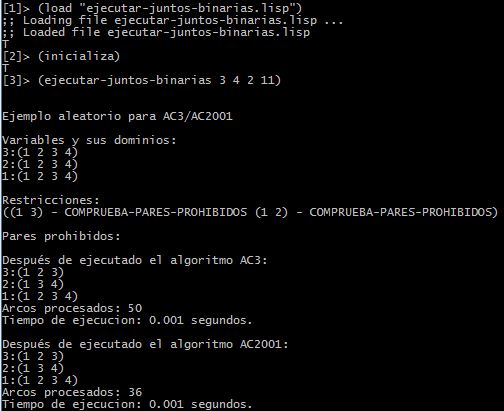


El parámetro nr (número de restricciones) irá variando y junto con la interfaz de salida se podrán recoger los datos y ofrecerlos en gráficos.

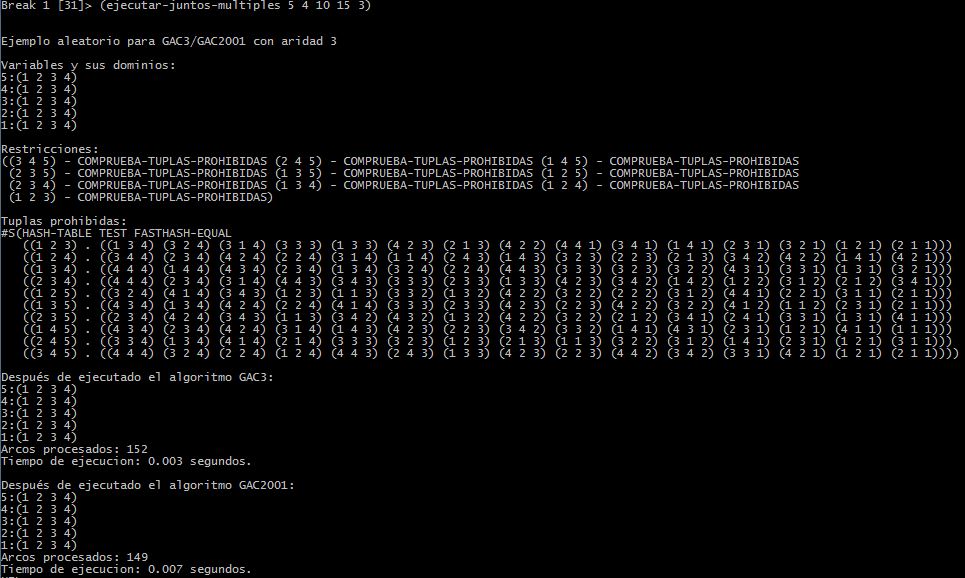
## **Ejecutar-juntos-binarias//multiples.lisp**

Con el fin de ejecutar el mismo ejemplo con restricciones binarias/múltiples en AC3 y AC2001 las sentencias serán (ejemplos):

* PSR binarios. Hay que comentar que para estos ejemplos hemos usado problemas muy pequeños, aunque más adelante probaremos con problemas más grandes.



* PSR de aridad n.



Hay que destacar de este ejemplo que, al ser muy poco restrictivo (al poner el ejemplo más restrictivo la imagen quedaba demasiado grande), no solo no quita ningún elemento del dominio de ninguna variable, si no que demuestra algo que explicaremos en las conclusiones, y es la poca mejora que supone el algoritmo AC-2001 en la gran mayoría de los casos.

# Conclusiones

Una vez terminados y probados ambos algoritmos, comprobamos que el algoritmo AC-2001 no supone apenas mejora para el AC-3 en la gran mayoría de los casos. La gran mejora que se incluye en el AC-2001 es la técnica de almacenar, como si de una memoria se tratase, los valores del dominio que ya ha comprobado para cada restricción, impidiendo así que, al tener que volver a comprobarlo vuelva a empezar desde el principio (modo en el que AC-3 opera)

Como probaremos en los ejemplos, esta mejora solamente se da en casos muy concretos de excesiva propagación (aunque sin llegar al extremo) donde un número elevado de restricciones y de tuplas prohibidas hace que el AC-2001 mejore sustancialmente los resultados del AC-3.

Usando los archivos *ac3-estadisticas.lisp* y *ac2001-estadisticas.lisp* hemos creado los siguientes gráficos que comparan ambos algoritmos para problemas.

Primero observamos que en ambos algoritmos tanto el tiempo como los arcos crecen de forma lineal conforme aumentan las restricciones:

* AC-3

|  |  |
| --- | --- |
|  |  |

|  |  |
| --- | --- |
|  |  |

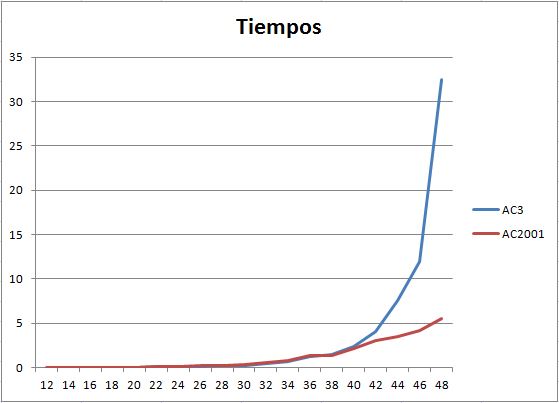
* AC-2001

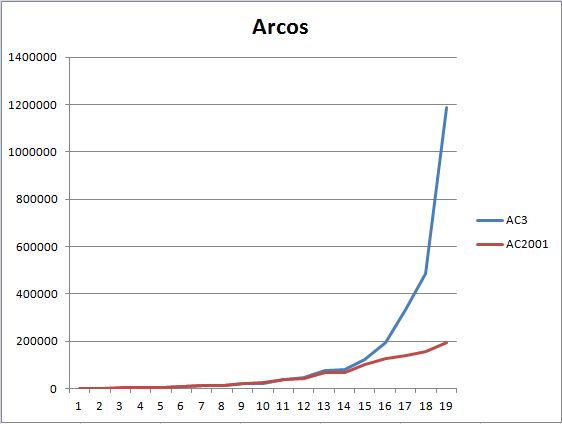
Luego comparamos ambos algoritmos con un dominio variable y con un número de pares prohibidos variable en función del dominio para que se notara la diferencia entre ambos algoritmos. Hemos de aclarar que hasta llegar a este gráfico tuvimos muchas pruebas en las que salían resultados prácticamente idénticos tanto en arcos analizados como en tiempo, y llegamos a este punto por el método de “prueba y error”, ya que los ejemplos que los autores exponen en su investigación tardaban demasiado en procesarse.

Al insertar por consola el siguiente método, y copiar los datos para generar el gráfico podemos observar que al ir aumentando el dominio del PSR, el algoritmo AC-2001 mejora sustancialmente al AC-3 en tiempo de ejecución y en arcos analizados:

(loop for nd from 12 to 48 by 2 do (ac3-estadisticas 20 nd 160 (\* 0.9 (\* nd nd))))

(loop for nd from 12 to 48 by 2 do (ac2001-estadisticas 20 nd 160 (\* 0.9 (\* nd nd))))

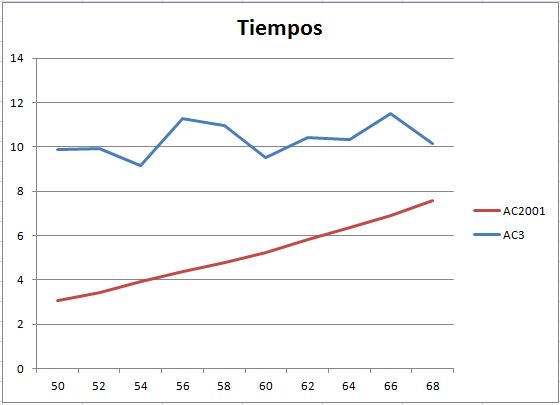




Otra cosa interesante que hemos notado respecto al algoritmo AC-2001, es que creíamos que mejoraba al AC-3 en los peores casos, pero que realmente no es así, ya que, al ir empeorando todavía más el ejemplo, observamos como ambos algoritmos van reduciendo sus diferencias tanto en tiempo de ejecución como en arcos analizados. Esto podemos verlo si introducimos el siguiente comando en la consola, analizando el tramo de dominio desde el último analizado para las gráficas anteriores.

(loop for nd from 50 to 68 by 2 do (ac3-estadisticas 20 nd 100 (\* 0.9 (\* nd nd))))

(loop for nd from 50 to 68 by 2 do (ac2001-estadisticas 20 nd 100 (\* 0.9 (\* nd nd))))





# Bibliografía

* **Apuntes de la asignatura Inteligencia Artificial 1**
* **“An optimal coarse-grained arc consistency algorithm”**
* <https://github.com/jaejin/99-lisp/blob/master/ninety-nine-lisp_P26.lisp>
* [Referencia de LISP](http://www.cs.cmu.edu/Groups/AI/html/cltl/clm/node1.html)

# ****Documentos adjuntos****

|  |  |
| --- | --- |
| Documento de referencia para algoritmo AC2001 |  |
| Tema 1 de Inteligencia Artificial 1 |  |
| Tema 5 de Inteligencia Artificial 1 |  |