Universitat d'Alacant
Universidad de Alicante

Memoria Práctica 1

Sistemas Inteligentes

Ingeniería Informática

Nombre: Juan Andrés Orocondo Álvarez

Fecha: 22/06/2024

1 Explicación de los algoritmos implementados

1.1 Forward Checking

El algoritmo de Forward Checking empieza cuando apretamos el botón FC, entonces realizará lo siguiente(Las funciones “imprime…” no las tendremos en cuenta ya que son funciones para realizar explicaciones)

```
if pulsaBotonFC(pos, anchoVentana, altoVentana):
    print("FC")
    imprimeAlmacen(almacen)
    if not ac3_executed:
        variables = identificarVariables(tablero, almacen)
        print("No AC3")
        imprimirVariables(variables)
    res = forwardChecking(tablero, variables, almacen)
    if not res:
        MessageBox.showwarning("Alerta", "No hay solución")
```

Primero explicaremos, las funciones auxiliares que utilizo.

```
def palabra_valida(tablero, fila, columna, orientacion, almacen):
    # Encuentra el inicio y el final de la palabra en la orientación especificada
    if orientacion == 'H':
        inicio = columna
        while inicio > 0 and tablero.getCelda(fila, inicio - 1) != LLENA:
            inicio -= 1
        final = columna
        while final < tablero.getAncho() - 1 and tablero.getCelda(fila, final + 1) != LLENA:
            final += 1
    else:
        inicio = fila
        while inicio > 0 and tablero.getCelda(inicio - 1, columna) != LLENA:
            inicio -= 1
        final = fila
        while final < tablero.getAlto() - 1 and tablero.getCelda(final + 1, columna) != LLENA:
            final += 1

    # Formar la palabra
    palabra = ''.join(tablero.getCelda(inicio + k, columna) if orientacion == 'V' else tablero.getCelda(fila, inicio + k) for k in range(final - inicio + 1))

    # Comprobar si la palabra formada es válida
    if '-' in palabra:  # Si hay celdas sin asignar en la palabra formada, ignorar
        return True
    return any(palabra == p for dom in almacen for p in dom.getLista() if dom.tam == len(palabra))
```

La función palabra_valida primeramente lo que hace es encontrar el inicio y el final de la palabra desde la columna en que se encuentra hasta que encuentre una celda llena o el borde del tablero.

Una vez hecho esto recopila las letras en las celdas que se han seleccionado para formar una palabra.

Finalmente se comprueba que si hay alguna celda con el carácter ‘-’ se asume que la palabra es válida y devuelve True, en cambio, si esto no ocurre compara la palabra formadas por las celdas con las palabras que hay en el almacén.

Esta función que acabamos de explicar se usa dentro de la siguiente:

```
def consistent(tablero, variable, palabra, almacen):
    for i, char in enumerate(palabra):
        # Establecer las coordenadas basadas en la orientación de la palabra
        fil, col = (variable.fila + i, variable.columna) if variable.orientacion == 'V' else (variable.fila, variable.columna + i)

        # Verificar que la letra actual encaje en el tablero
        if tablero.getCelda(fil, col) not in [VACIA, char]:
            return False

        # Verificación de palabras horizontales y verticales para cada letra
        if not palabra_valida(tablero, fil, col, 'H', almacen):
            return False
        if not palabra_valida(tablero, fil, col, 'V', almacen):
            return False

    return True
```

La función consistent verifica si una palabra puede o no ser asignada a una variable sin violar las restricciones del crucigrama.

Primero establece las coordenadas basadas en la orientación de la palabra, luego verifica que la letra que se está intentando añadir al crucigrama encaje, ya sea porque esta vacía o porque en esa celda está esa misma letra.

Y finalmente se comprueba que al añadir esa letra, esta afecta a la validez de las palabras cruzadas que se forman tanto en horizontal como en vertical(palabra_valida).

A continuación, tenemos la función identificarVariables, se crea con el objetivo de buscar dentro del almacen todas las posibles variables que se puedan formar tanto en horizontal como en vertical en el tablero.

```
def identificarVariables(tablero, almacen):
    variables = []
    var_id = 0

    # Horizontal variables
    for fil in range(FILS):
        col = 0
        while col < COLS:
            # Comienza si la celda está vacía o contiene una letra
            if tablero.getCelda(fil, col) == VACIA or tablero.getCelda(fil, col).isalpha():
                start_col = col
                valid = True
                palabra = ""

                # Extiende hacia la derecha mientras las condiciones sean apropiadas
                while col < COLS and (tablero.getCelda(fil, col) == VACIA or tablero.getCelda(fil, col).isalpha()):
                    palabra += tablero.getCelda(fil, col) if tablero.getCelda(fil, col) != VACIA else ' '
                    col += 1
```

```
    # Verifica la longitud de la palabra y si es válida
    if col - start_col > 1:
        var = Variable(f"V{var_id}", fil, start_col, col - start_col, 'H')
        pos = busca(almacen, col - start_col)
        if pos != -1:
            # Filtra dominio para incluir solo palabras que coincidan con las letras introducidas
            filtered_domain = [word for word in almacen[pos].getLista() if all((c == ' ' or c == word[i]) for i, c in enumerate(palabra) if i < len(word))]
            if filtered_domain:
                var.setDominio(filtered_domain)
                variables.append(var)
                var_id += 1
col += 1
```

```
# Vertical variables
for col in range(COLS):
    fil = 0
    while fil < FILS:
        if tablero.getCelda(fil, col) == VACIA or tablero.getCelda(fil, col).isalpha():
            start_fil = fil
            valid = True
            palabra = ""

            # Extiende hacia abajo mientras las condiciones sean apropiadas
            while fil < FILS and (tablero.getCelda(fil, col) == VACIA or tablero.getCelda(fil, col).isalpha()):
                palabra += tablero.getCelda(fil, col) if tablero.getCelda(fil, col) != VACIA else ' '
                fil += 1
```

```
        # Verifica la longitud de la palabra y si es válida
        if fil - start_fil > 1:
            var = Variable(f"V{var_id}", start_fil, col, fil - start_fil, 'V')
            pos = busca(almacen, fil - start_fil)
            if pos != -1:
                # Filtra dominio para incluir solo palabras que coincidan con las letras introducidas
                filtered_domain = [word for word in almacen[pos].getLista() if all((c == ' ' or c == word[i]) for i, c in enumerate(palabra) if i < len(word))]
                if filtered_domain:
                    var.setDominio(filtered_domain)
                    variables.append(var)
                    var_id += 1
    fil += 1

return variables
```

Esta función recorre cada celda del tablero y en cuanto encuentra una celda vacía empieza a contar la longitud de la palabra hasta que encuentra una celda vacía, si la longitud de esta es mayor que 1 entonces creará un objeto de tipo variable y lo añadirá a la lista de variables.

Para cada variable identificada buscara dentro de nuestro diccionario palabras que coincidan con la longitud de esta, además, si esta variable contiene una letra en alguna celda, buscará dentro del diccionario palabras que tengan esa letra en esa posición.

Obviamente estas comprobaciones se harán para las variables tanto horizontales, como verticales.

Ahora, tenemos las últimas dos funciones auxiliares que son assign y unassign:

```
def assign(tablero, variable, palabra):
    for i, char in enumerate(palabra):

        if variable.orientacion == 'H':
            tablero.setCelda(variable.fila, variable.columna + i, char)
        else:
            tablero.setCelda(variable.fila + i, variable.columna, char)
```

```
def unassign(tablero, variable, palabra):
    for i, char in enumerate(palabra):
        if variable.orientacion == 'H':
            tablero.setCelda(variable.fila, variable.columna + i, VACIA)
        else:
            tablero.setCelda(variable.fila + i, variable.columna, VACIA)
```

Como su propio nombre indica usaremos estas funciones para asignar o desasignar una palabra del crucigrama. Se comprueba la orientación en la que se va a insertar esta palabra y se añade al crucigrama.

Y, finalmente, la función principal forwardChecking:

```
def forwardChecking(tablero, variables, almacen):
    if not variables:
        return True
    variable = variables[0]
    for palabra in variable.dominio:
        if consistent(tablero, variable, palabra, almacen):
            assign(tablero, variable, palabra)
            if forwardChecking(tablero, variables[1:], almacen):
                return True
            unassign(tablero, variable, palabra)
    return False
```

Como podemos observar, lo primero que comprueba la función es que la lista de variables no esté vacía en caso de que lo esté, se habrá encontrado una solución valida al crucigrama.

La función selecciona la primera variable de la lista y comprueba cada palabra en su dominio, si la palabra es consistente entonces procederá a asignar esta palabra al crucigrama y realizará la llamada recursiva a forwardChecking con el resto de la lista de variables, en caso de que esta no acabe encontrando una solución devolverá false y se desasignará la palabra y se probará con la siguiente palabra que hay en la variable.

1.2 AC3

Como hicimos en el anterior algoritmo, primero explicaremos las funciones auxiliares que utilizamos para realizar el algoritmo de AC3.

```
def is_arc_consistent(tablero, variable1, palabra1, variable2, palabra2, almacen):
    if variable1.orientacion == 'H':
        cells1 = [(variable1.fila, variable1.columna + i) for i in range(variable1.longitud)]
    else:
        cells1 = [(variable1.fila + i, variable1.columna) for i in range(variable1.longitud)]

    if variable2.orientacion == 'H':
        cells2 = [(variable2.fila, variable2.columna + i) for i in range(variable2.longitud)]
    else:
        cells2 = [(variable2.fila + i, variable2.columna) for i in range(variable2.longitud)]

    for (fil1, col1) in cells1:
        if (fil1, col1) in cells2:
            i = cells1.index((fil1, col1))
            j = cells2.index((fil1, col1))
            if palabra1[i] != palabra2[j]:
                return False

    return True
```

Primeramente, esta función identifica las celdas ocupadas por las variables1 y variable2.

Comprueba si estas celdas se cruzan en algún punto del tablero, si es así comprueba que la celda en la que coinciden contenga la misma letra en caso contrario esta devuelve false.

El objetivo de esta función es asegurar que las asignaciones no causen conflictos en el tablero, especialmente donde dos variables se intersectan.

Revise, el objetivo de esta función es garantizar que todas las asignaciones posibles para una variable sean consistentes con respecto a otra variable dada:

```
def revise(tablero, vi, vj, almacen):
    revised = False
    for x in vi.dominio[:]:
        if not any(is_arc_consistent(tablero, vi, x, vj, y, almacen) for y in vj.dominio):
            vi.dominio.remove(x)
            revised = True
    return revised
```

Esta función itera sobre cada valor ’x’ en el dominio de vi, para cada valor de ‘x’ comprueba que exista al menos un valor dentro del dominio de vj tal que estas sean consistentes(is_arc_consistent), si no existe procederá a borrar ese valor x del dominio vi.

Finalmente, esta función devolverá True si se ha cambiado el dominio de vi.

are_neighbors, esta función comprueba si dos variables comparten al menos una celda:

```
def are_neighbors(vi, vj):
    # Determina si dos variables son vecinos (comparten al menos una celda)
    if vi.orientacion == 'H':
        cells_vi = [(vi.fila, vi.columna + i) for i in range(vi.longitud)]
    else:
        cells_vi = [(vi.fila + i, vi.columna) for i in range(vi.longitud)]

    if vj.orientacion == 'H':
        cells_vj = [(vj.fila, vj.columna + i) for i in range(vj.longitud)]
    else:
        cells_vj = [(vj.fila + i, vj.columna) for i in range(vj.longitud)]

    return bool(set(cells_vi) & set(cells_vj))
```

Por último, la función AC3:

```python
def ac3(tablero, variables, almacen):
    print("DOMINIOS ANTES DEL AC3")
    for variable in variables:
        print(f"Nombre {variable.nombre} Posición {variable.fila} {variable.columna} Tipo: {variable.orientacion} Dominio: {variable.dominio}")
    # Inicializa la cola de arcos con todas las aristas del grafo de restricciones
    queue = [(vi, vj) for vi in variables for vj in variables if vi != vj and are_neighbors(vi, vj)]
    while queue:
        (vk, vm) = queue.pop(0)
        cambio = False
        if revise(tablero, vk, vm, almacen):
            if not vk.dominio:
                return False  # Si el dominio se queda vacío, no hay solución
            cambio = True
        if cambio:
            for vr in variables:
                if vr != vk and are_neighbors(vr, vk):
                    queue.append((vr, vk))
    print("\nDOMINIOS DESPUÉS DEL AC3")
    for variable in variables:
        print(f"Nombre {variable.nombre} Posición {variable.fila} {variable.columna} Tipo: {variable.orientacion} Dominio: {variable.dominio}")
    return True
```

Como podemos observar la función imprime tanto al principio como al final del algoritmo los dominios para poder observar como se han reducido las variables de estos.

A parte de esto, primeramente, inicializa la cola con grupos de variables que sean vecinos ya que solo estos deberán de ser revisados para ver si son consistentes.

Cuando entramos dentro de la recursión vemos que extrae un par de variables con las que llama a la funcion revise y, si se hizo alguna reducción se cambiara el valor de cambio a True, en cambio, si este dominio se queda vacío devolveremos False, ya que no habrá solución.

Luego si se cambio el dominio se añadirán a la cola todos los pares que involucren a la primera variable junto con sus vecinos, para asegurar que los dominios se mantienen consistentes con todas las restricciones.

2 Explicación detallada de un problema pequeño

El problema que vamos a utilizar para explicar el forward checking será el siguiente:

Tendremos que, en las posiciones (0,2), (1,2), (2,0) celdas negras y en la posición (2,1) la letra A y en la posición (0,0) la letra L.

El diccionario que utilizaremos será el siguiente:

```
Esto es un
ejemplo de fichero
de prueba
para la practica
con varias palabras de distinta longitud como totem osera l a b no
rosa olor lal rol ron ola sol ara lala ar pero lotero retos setos
osos lotera pera romano esopo romana
```

Una vez apretemos el botón de FC nos mostrará el siguiente diccionario: ordenado por longitud de variables:

```
4
ESTO PARA COMO ROSA OLOR LALA PERO OSOS PERA
2
ES UN DE LA NO AR
7
EJEMPLO FICHERO
6
PRUEBA VARIAS LOTERO LOTERA ROMANO ROMANA
8
PRACTICA PALABRAS DISTINTA LONGITUD
3
CON LAL ROL RON OLA SOL ARA
5
TOTEM OSERA RETOS SETOS ESOPO
1
L A B
```

También hay que tener en cuenta las restricciones que hay por coincidencias de celdas, colocaré las de mayor importancia ya que tienen algo que ver con las letras que se introducen en el crucigrama antes de resolverlo y serían las siguientes:

Variables V0 y V3

(0,0,'H'), (0,0,'V'), (0,0,'L')

Variables V0 y V4

(0,0,'H'), (0,1,'V'), (0,1,'-')

Variables V4 y V2

(0,1,'V'), (2,1,'H'), (2,1,'A')

El resto de las variables por coincidencia de celdas son celdas vacías.

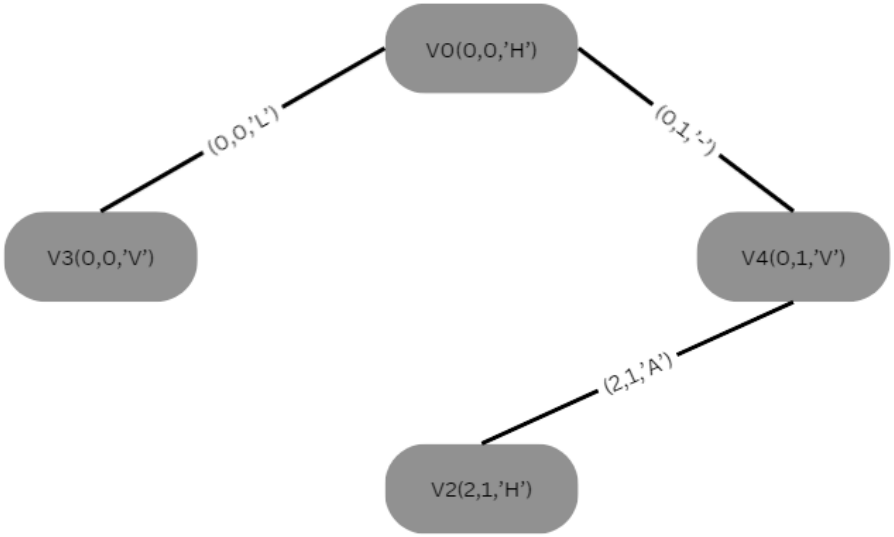
Y una vez revisadas las restricciones que se mencionan anteriormente obtendremos las siguientes variables que se usarán para resolver el crucigrama:

```
Variables actuales en el tablero:
Nombre: V0
Posición: (0, 0)
Longitud: 2
Orientación: H
Dominio: ['LA']
-----
Nombre: V1
Posición: (1, 0)
Longitud: 2
Orientación: H
Dominio: ['ES', 'UN', 'DE', 'LA', 'NO', 'AR']
```

```
Nombre: V2
Posición: (2, 1)
Longitud: 2
Orientación: H
Dominio: ['AR']
-----
Nombre: V3
Posición: (0, 0)
Longitud: 2
Orientación: V
Dominio: ['LA']
-----
Nombre: V4
Posición: (0, 1)
Longitud: 3
Orientación: V
Dominio: ['OLA', 'ARA']
```

3 Grafo de restricciones del problema

El grafo resultante, donde los nodos son las variables y las aristas indicarían las restricciones que tienen entre los nodos, de las restricciones sería el siguiente:



4 Traza del algoritmo FC

```
4
ESTO PARA COMO ROSA OLOR LALA PERO OSOS PERA
2
ES UN DE LA NO AR
7
EJEMPLO FICHERO
6
PRUEBA VARIAS LOTERO LOTERA ROMANO ROMANA
8
PRACTICA PALABRAS DISTINTA LONGITUD
3
CON LAL ROL RON OLA SOL ARA
5
TOTEM OSERA RETOS SETOS ESOPO
1
```

Como se ha visto anteriormente este es el diccionario del que disponemos.

```
Variables actuales en el tablero:
Nombre: V0
Posición: (0, 0)
Longitud: 2
Orientación: H
Dominio: ['LA']
-----
Nombre: V1
Posición: (1, 0)
Longitud: 2
Orientación: H
Dominio: ['ES', 'UN', 'DE', 'LA', 'NO', 'AR']
-----
Nombre: V2
Posición: (2, 1)
Longitud: 2
Orientación: H
Dominio: ['AR']
-----
Nombre: V3
Posición: (0, 0)
Longitud: 2
Orientación: V
Dominio: ['LA']
-----
Nombre: V4
Posición: (0, 1)
Longitud: 3
Orientación: V
Dominio: ['OLA', 'ARA']
```

Y este seria el resultado después de ejecutar la función identificarVariables para reducir el diccionario y usar así únicamente palabras validas dentro de nuestro crucigrama.

Iniciamos forwardChecking

Dominios de las variables no asignadas al inicio:

V0: ['LA']

V1: ['ES', 'UN', 'DE', 'LA', 'NO', 'AR']

V2: ['AR']

V3: ['LA']

V4: ['OLA', 'ARA']

Variable seleccionada: V0

Asignamos "LA" a la variable seleccionada.

Recorremos todas las variables sin asignar y eliminamos "LA" del resto de dominios para no permitir palabras repetidas.

Hacemos la llamada recursiva a forwardChecking

Como no crea ningún conflicto con otras palabras seguimos adelante

Dominios de las variables no asignadas al inicio:

V1: ['ES', 'UN', 'DE', 'LA', 'NO', 'AR']

V2: ['AR']

V3: ['LA']

V4: ['OLA', 'ARA']

Variable seleccionada: V1

Asignamos "ES" a la variable seleccionada.

Recorremos todas las variables sin asignar y eliminamos "ES" del resto de dominios para no permitir palabras repetidas.

Hacemos la llamada recursiva a forward_checking

Dominios de las variables no asignadas al inicio:

V2: ['AR']

V3: ['LA']

V4: ['OLA', 'ARA']

Como 'ES' no se puede añadir porque crea un conflicto con la letra 'A' en la posición (2,1) se restaura.

Dominios de las variables no asignadas después de restaurar:

V1: ['ES', 'UN', 'DE', 'LA', 'NO', 'AR']

V2: ['AR']

V3: ['LA']

V4: ['OLA', 'ARA']

Variable seleccionada: V1

Asignamos "UN" a la variable seleccionada.

Recorremos todas las variables sin asignar y eliminamos "UN" del resto de dominios para no permitir palabras repetidas.

Hacemos la llamada recursiva a forward_checking

Dominios de las variables no asignadas al inicio:

V2: ['AR']

V3: ['LA']

V4: ['OLA', 'ARA']

Como 'UN' no se puede añadir porque crea un conflicto con la letra 'A' en la posición (2,1) se restaura.

Dominios de las variables no asignadas después de restaurar:

V1: ['ES', 'UN', 'DE', 'LA', 'NO', 'AR']

V2: ['AR']

V3: ['LA']

V4: ['OLA', 'ARA']

Variable seleccionada: V1

Asignamos "DE" a la variable seleccionada.

Recorremos todas las variables sin asignar y eliminamos "DE" del resto de dominios para no permitir palabras repetidas.

Hacemos la llamada recursiva a forward_checking

Dominios de las variables no asignadas al inicio:

V2: ['AR']

V3: ['LA']

V4: ['OLA', 'ARA']

Como 'DE' no se puede añadir porque crea un conflicto con la letra 'A' en la posición (2,1) se restaura.

Dominios de las variables no asignadas después de restaurar:

V1: ['ES', 'UN', 'DE', 'LA', 'NO', 'AR']

V2: ['AR']

V3: ['LA']

V4: ['OLA', 'ARA']

Variable seleccionada: V1

Asignamos "LA" a la variable seleccionada.

Recorremos todas las variables sin asignar y eliminamos "LA" del resto de dominios

para no permitir palabras repetidas.

Hacemos la llamada recursiva a forward_checking

Dominios de las variables no asignadas al inicio:

V2: ['AR']

V3: ['LA']

V4: ['OLA', 'ARA']

Como 'LA' no se puede añadir porque crea un conflicto con la letra 'A' en la posición (2,1) se restaura.

Dominios de las variables no asignadas después de restaurar:

V1: ['ES', 'UN', 'DE', 'LA', 'NO', 'AR']

V2: ['AR']

V3: ['LA']

V4: ['OLA', 'ARA']

Variable seleccionada: V1

Asignamos "NO" a la variable seleccionada.

Recorremos todas las variables sin asignar y eliminamos "NO" del resto de dominios

para no permitir palabras repetidas.

Hacemos la llamada recursiva a forward_checking

Dominios de las variables no asignadas al inicio:

V2: ['AR']

V3: ['LA']

V4: ['OLA', 'ARA']

Como 'NO' no se puede añadir porque crea un conflicto con la letra 'A' en la posición (2,1) se restaura.

Dominios de las variables no asignadas después de restaurar:

V1: ['ES', 'UN', 'DE', 'LA', 'NO', 'AR']

V2: ['AR']

V3: ['LA']

V4: ['OLA', 'ARA']

Variable seleccionada: V1

Asignamos "AR" a la variable seleccionada.

Recorremos todas las variables sin asignar y eliminamos "AR" del resto de dominios para no permitir palabras repetidas.

Hacemos la llamada recursiva a forward_checking

Dominios de las variables no asignadas al inicio:

V2: ['AR']

V3: ['LA']

V4: ['OLA', 'ARA']

Variable seleccionada: V2

Asignamos "AR" a la variable seleccionada.

Recorremos todas las variables sin asignar y eliminamos "AR" del resto de dominios para no permitir palabras repetidas.

Hacemos la llamada recursiva a forward_checking

Dominios de las variables no asignadas al inicio:

V3: ['LA']

V4: ['OLA', 'ARA']

Variable seleccionada: V3

Asignamos "LA" a la variable seleccionada.

Recorremos todas las variables sin asignar y eliminamos "LA" del resto de dominios para no permitir palabras repetidas.

Hacemos la llamada recursiva a forward_checking

Dominios de las variables no asignadas al inicio:

V4: ['OLA', 'ARA']

Variable seleccionada: V4

Asignamos "ARA" a la variable seleccionada, ya que OLA no sería consistente con el crucigrama.

Recorremos todas las variables sin asignar y eliminamos "ARA" del resto de dominios para no permitir palabras repetidas.

En este momento la lista de variables se habría quedado vacía así que el algoritmo habría encontrado una solución, en este caso sería esta:

5 Traza del algoritmo AC3

```
V0: ['LA']
V1: ['ES', 'UN', 'DE', 'LA', 'NO', 'AR']
V2: ['AR']
V3: ['LA']
V4: ['OLA', 'ARA']
```

Al igual que antes, empezamos con estas variables y iniciaremos el algoritmo AC3 para comprobar la consistencia de las palabras entre las diferentes variables.

Revisando arco entre V0 y V3 porque deben ser consistentes según las restricciones.

Como al menos una variable de V3 es consistente con las variables de V0 pasa a la siguiente.

Revisando arco entre V0 y V4 porque deben ser consistentes según las restricciones.

Como al menos una variable de V4 es consistente con las variables de V0 pasa a la siguiente.

Revisando arco entre V1 y V3 porque deben ser consistentes según las restricciones.

Eliminando ES de V1 porque no es consistente con ningún valor en el dominio de V3

Eliminando UN de V1 porque no es consistente con ningún valor en el dominio de V3

Eliminando DE de V1 porque no es consistente con ningún valor en el dominio de V3

Eliminando LA de V1 porque no es consistente con ningún valor en el dominio de V3

Eliminando NO de V1 porque no es consistente con ningún valor en el dominio de V3

Añadiendo arco (V3, V1) a la cola porque cambiar el dominio de V1 podría afectar a V3

Añadiendo arco (V4, V1) a la cola porque cambiar el dominio de V1 podría afectar a V4

Revisando arco entre V1 y V4 porque deben ser consistentes según las restricciones.

Como al menos una variable de V4 es consistente con las variables de V1 pasa a la siguiente.

Revisando arco entre V2 y V4 porque deben ser consistentes según las restricciones.

Como al menos una variable de V4 es consistente con las variables de V2 pasa a la siguiente.

Revisando arco entre V3 y V0 porque deben ser consistentes según las restricciones.

Como al menos una variable de V0 es consistente con las variables de V3 pasa a la siguiente.

Revisando arco entre V3 y V1 porque deben ser consistentes según las restricciones.

Como al menos una variable de V1 es consistente con las variables de V3 pasa a la siguiente.

Revisando arco entre V4 y V0 porque deben ser consistentes según las restricciones.

Eliminando OLA de V4 porque no es consistente con ningún valor en el dominio de V0

Añadiendo arco (V0, V4) a la cola porque cambiar el dominio de V4 podría afectar a V0

Añadiendo arco (V1, V4) a la cola porque cambiar el dominio de V4 podría afectar a V1

Añadiendo arco (V2, V4) a la cola porque cambiar el dominio de V4 podría afectar a V2

Revisando arco entre V4 y V1 porque deben ser consistentes según las restricciones.

Como al menos una variable de V1 es consistente con las variables de V4 pasa a la siguiente.

Revisando arco entre V4 y V2 porque deben ser consistentes según las restricciones.

Como al menos una variable de V2 es consistente con las variables de V4 pasa a la siguiente.

Revisando arco entre V3 y V1 porque deben ser consistentes según las restricciones.

Como al menos una variable de V1 es consistente con las variables de V3 pasa a la siguiente.

Revisando arco entre V4 y V1 porque deben ser consistentes según las restricciones.

Como al menos una variable de V1 es consistente con las variables de V4 pasa a la siguiente.

Revisando arco entre V0 y V4 porque deben ser consistentes según las restricciones.

Como al menos una variable de V4 es consistente con las variables de V0 pasa a la siguiente.

Revisando arco entre V1 y V4 porque deben ser consistentes según las restricciones.

Como al menos una variable de V4 es consistente con las variables de V1 pasa a la siguiente.

Revisando arco entre V2 y V4 porque deben ser consistentes según las restricciones.

Como al menos una variable de V4 es consistente con las variables de V2 pasa a la siguiente.

Ya no quedan mas pares de arcos en la cola así que AC3 finaliza y, como la cola ha quedado vacía, obtenemos el siguiente resultado:

AC3 Final: Dominios después de AC3

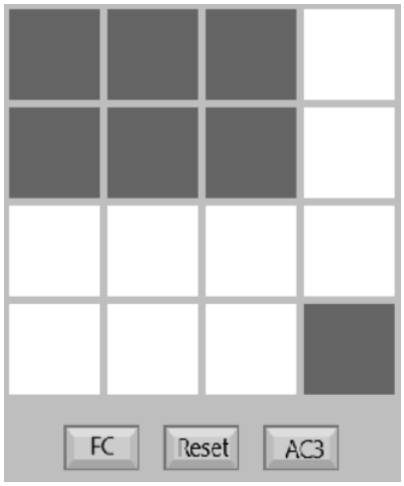
V0: ['LA'] V1: ['AR']

V2: ['AR'] V3: ['LA']

V4: ['ARA']

6 Sección de experimentación

Matrices 4x4

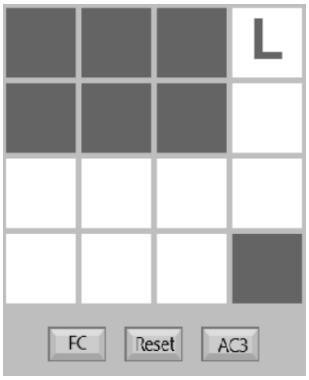


En esta matriz, usando el mismo diccionario que se usó anteriormente obtenemos el siguiente resultado de AC3:

```
AC3 Final: Dominios después de AC3
V0: ['LALA']
V1: ['ARA']
V2: ['LA']
V3: ['AR']
V4: ['LA']
V5: ['OLA', 'ARA']
```

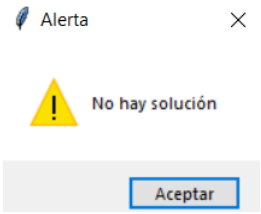
Y, luego finalmente al hacer FC obtenemos el siguiente resultado:

Ahora, cambiamos ligeramente el crucigrama anterior para ver si obtenemos un resultado.



El único cambio que hicimos es añadir una L en la posición (0,3)

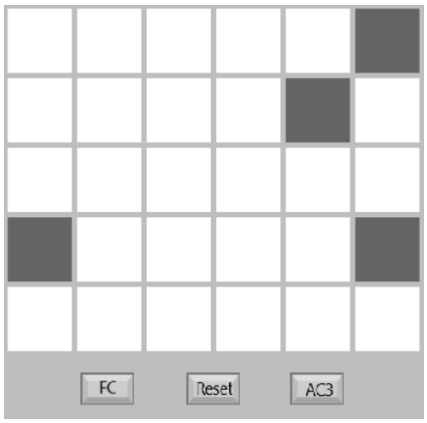
Y esto es lo que sucede cuando intentamos hacer FC:



Nos sale la alerta de que este crucigrama no tiene solución, lo mismo sucede cuando intentamos hacer el algoritmo de AC3, nos vuelve a salir la misma alerta.

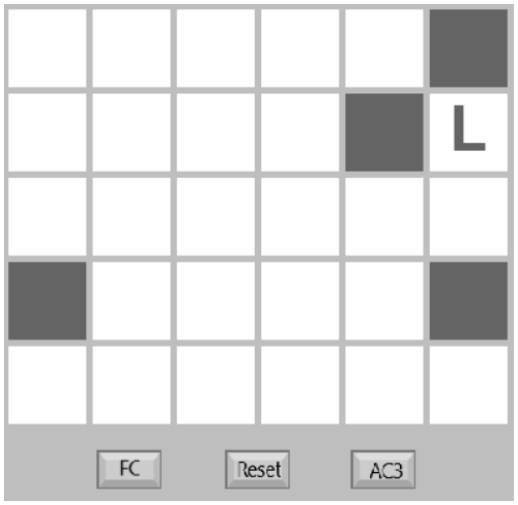
Matriz 5x6

Un caso curioso es ver como el algoritmo se adapta a las diferentes restricciones que le colocamos.



En este caso cuando iniciamos el FC nuestro algoritmo saca el siguiente resultado:

Un resultado, completamente válido y que resuelve el crucigrama, pero cuando introducimos una letra cambian dos palabras de este crucigrama.



Ahora probaremos con este crucigrama, que simplemente le añadimos una letra y observaremos los resultados del algoritmo FC.

Como podemos observar, cambiaron dos palabras, lotera(2,0,'H') y la(1,5,'V') con respecto al resultado anterior.

Vamos a probar ahora con el algoritmo AC3.

Cuando realizamos la prueba con la matriz sin la restricción de la letra obtenemos el siguiente resultado:

```
AC3 Final: Dominios después de AC3
V0: ['RETOS', 'SETOS']
V1: ['OSOS']
V2: ['LOTERO', 'LOTERA']
V3: ['PERO']
V4: ['ROMANO', 'ROMANA']
V5: ['ROL', 'SOL']
V6: ['ESOPO']
V7: ['TOTEM']
V8: ['OSERA']
V9: ['RON']
V10: ['LA', 'NO']
```

Como podemos observar, se añaden tanto lotera como lotero y también la y no en las variables que crean luego conflicto, cuando añadimos la letra L.

Cuando añadimos dicha letra al crucigrama obtendríamos el siguiente resultado:

```
AC3 Final: Dominios después de AC3
V0: ['RETOS', 'SETOS']
V1: ['OSOS']
V2: ['LOTERA']
V3: ['PERO']
V4: ['ROMANO', 'ROMANA']
V5: ['ROL', 'SOL']
V6: ['ESOPO']
V7: ['TOTEM']
V8: ['OSERA']
V9: ['RON']
V10: ['LA']
```

Claramente se ve que el algoritmo decidió eliminar las palabras lotero y no ya que estas al tener la letra L en la posición (1,5) crean conflictos.

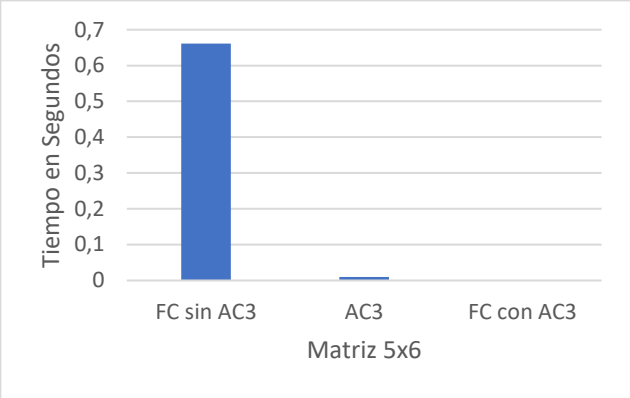
7 Estudio de tiempos de los algoritmos

Vamos a realizar diferentes pruebas en las que usaremos crucigramas de diferentes tamaños vistos a lo largo de la memoria, pero con el mismo diccionario de palabras.

Todas estas pruebas se han realizado con crucigramas que tienen solución y sin añadir letras como restricciones iniciales.

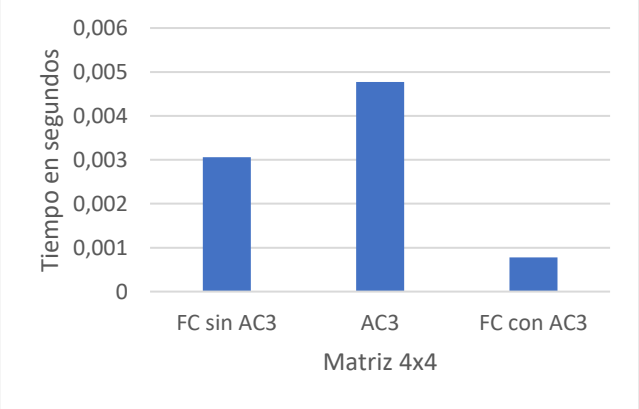
Matriz 5x6

Algoritmo	Tiempo (Segundos)
FC sin AC3	0.66164
AC3	0.00977
FC con AC3	0.00078



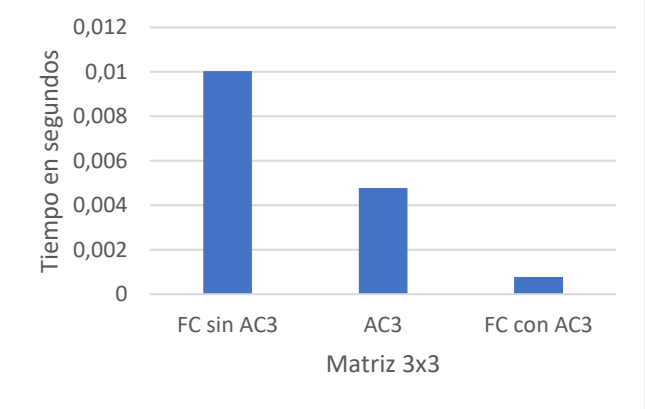
Matriz 4x4

Algoritmo	Tiempo (Segundos)
FC sin AC3	0.00306
AC3	0.00477
FC con AC3	0.00078



Matriz 3x3

Algoritmo	Tiempo (Segundos)
FC sin AC3	0.01003
AC3	0.00478
FC con AC3	0.00078



Como podemos observar, las matrices de 3x3 y 4x4 tardan un tiempo muy pequeño, esta puede ser una de las razones por las que en la matriz 4x4 tarde mas en resolverse el AC3 que el FC. Cuando ya pasamos a tamaños de crucigramas algo mas grandes y menos

restrictivas por las casillas llenas observamos que el tiempo que tarda en resolverse el algoritmo FC escala mucho más.

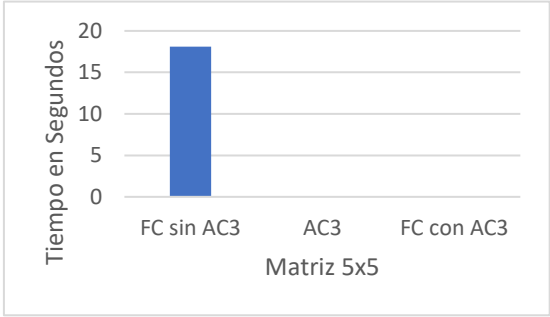
Ahora haremos las mismas pruebas, pero añadiendo más palabras a nuestro diccionario y analizaremos los resultados.

```
Esto es un
ejemplo de fichero
de prueba
para la practica
con varias palabras de distinta longitud como totem osera l a b no
rosa olor lal rol ron ola sol ara lala ar pero lotero retos setos
osos lotera pera romano esopo romana
perro gato elefante tigre león zorro quetzal rinoceronte jabalí
hormiga abeja caballo gorila oruga mantis mariposa alce topo
ciervo nutria suricata iguana alacrán tarántula camello
dromedario ardilla mapache tejón marmota
halcón cuervo paloma cisne loro canario tucán pavo ganso
cisne lobo zorro canguro koala puma ocelote hiena elefante
hipopótamo rinoceronte búfalo panda antílope ciervo alce
castor nutria morsa foca delfín ballena tiburón orca
```

Este será el diccionario que utilizaremos ahora para realizar las pruebas.

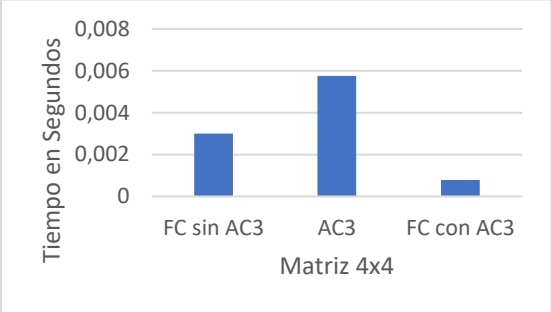
Matriz 5x6

Algoritmo	Tiempo (Segundos)
FC sin AC3	18.10193
AC3	0.01836
FC con AC3	0.00101



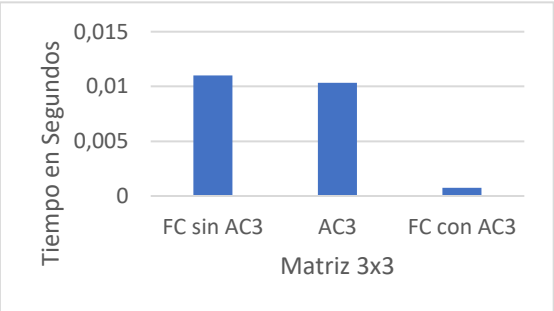
Matriz 4x4

Algoritmo	Tiempo (Segundos)
FC sin AC3	0.003
AC3	0.00575
FC con AC3	0.00078



Matriz 3x3

Algoritmo	Tiempo (Segundos)
FC sin AC3	0.01101
AC3	0.01034
FC con AC3	0.00076



Como podemos observar en las matrices de 3x3 y 4x4 no hay un cambio realmente significativo, en cambio el principal cambio es el tiempo que tarda en realizar FC en la matriz de 5x6 que pasa a tardar 18 segundos por tener que ir consultando mayores números de palabras.

8 Bibliografía

Para la comprensión del algoritmo de FC a parte del material dado en clase, recurrí a las siguientes páginas:

http://opac.pucv.cl/pucv_txt/txt-7500/UCF7877_01.pdf

https://www.cs.us.es/~fsancho/Cursos/SVRAI/CSP.md.html

Y para comprender mejor el algoritmo AC3:

https://www.toolify.ai/es/ai-news-es/satisfaccin-de-restricciones-algoritmo-ac3-2042502

También use ChatGPT3.5/4/4o para obtener más explicaciones del funcionamiento de los algoritmos, generación de código, generar impresiones por terminal para encontrar los posibles problemas que he ido teniendo y resolución de problemas menores.