Práctica 5: CSP

Herramienta utilizada: Conflex

Pascual Andrés Carrasco Gómez

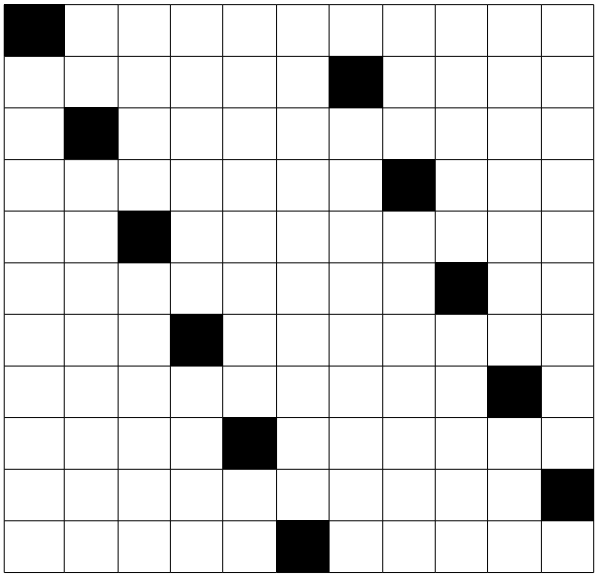
Índice

Parte 1: N-Reinas

En este apartado de la práctica vamos a comentar los resultados que hemos obtenido al ejecutar diferentes instancias del problema de las n reinas, las instancias que hemos escogido son para 4, 6, 9, 10, 11 y 12 reinas. El valor que hemos medido es el número de instancias que devuelve conflex ya que el tiempo de cálculo es dependiente de cada procesador.

(1 solución)	n=4	n=6	n=9	n=10	n=11	n=12?
BT	26	171	333	975	517	3066
FC	7	18	20	47	28	104
RFLA	5	18	17	42	20	79

En esta tabla hemos realizado ejecuciones obteniendo siempre una solución variando el algoritmo de búsqueda, los algoritmos de búsqueda utilizados son backtracking (BT), forward-checking (FC), looking-ahead (RFLA), como ya sabemos RFLA es el que mas informado esta para realizar la búsqueda, después viene FC y por último el menos informado es BT, esto lo vemos reflejado claramente en los resultados obtenidos en la tabla mediante el número de instancias generadas para obtener la solución. Observamos que como es comprensible como va incrementado n necesitamos mas número de instancias para encontrar la solución pero esto no lo podemos garantizar y lo vemos con n = 11 reinas donde el número de instancias para los tres algoritmos realizados es inferior respecto con n = 10 reinas, esto se debe a que el tablero con 11 reinas tiene una configuración para resolver el problema mas trivial que para n = 10 reinas.



La solución que observamos en la ilustración de la izquierda corresponde a la primera solución obtenida para el problema de las n reinas con n = 11 reinas, al ver la solución de forma gráfica podemos entender que se necesitan menos búsqueda en el árbol que para 10 reinas donde la solución no es tan trivial.

(todas las soluciones)	n=4	n=6	n=9	n=10	n=11	n=12?
BT	60	894	72378	348150	1806706	**-------**
FC	14	86	4741	17048	75147	**-------**
RFLA	10	74	3809	12840	55671	**-------**

En esta tabla hemos realizado ejecuciones obteniendo todas las soluciones que existen para el problema de las n reinas con las n especificadas anteriormente y variando el algoritmo de búsqueda como en el apartado anterior, en los resultados obtenidos en esta tabla podemos observar mejor que la resolución de este tipo de problemas es NP, vemos que el coste del problema va aumentando de forma exponencial a medida que incrementa n, con n = 12 el problema explota y después de 20 minutos de ejecución aun no nos ha devuelto la solución y hemos finalizado la ejecución.

Heuristicas variables (1 solución) con <u>FC</u> (elegid BT, FC o RFLA para esta prueba)	n=4	n=6	n=9	n=10	n=11	n=12?
Static: smallest_domain	7	18	20	47	28	104
Static: smallest_domain_by_degree	7	18	20	47	28	104
Dynamic: smallest_domain	6	18	23	22	39	86
Dynamic: smallest_domain_by_deg	5	19	23	44	232	83

En esta tabla hemos escogido forward-checking (FC) como algoritmo de búsqueda para realizar las ejecuciones, nos hemos basado en obtener una solución y hemos variado los parámetros para heurísticas de variables tanto estáticas como dinámicas, tanto en el apartado de estáticas como dinámicas hemos estudiado dos opciones:

- smallest_domain: Por orden del dominio mas pequeño.
- smallest_domain_by_degree: Relación talla_dominio/grado mas pequeño.

Observamos que con heurísticas estáticas no tenemos mejora respecto a la primera tabla que hemos completado ya que se necesitan el mismo número de instancias en ambos casos.
En el apartado de heurísticas dinámicas observamos que para n = 4,12 los resultados mejoran siendo mejor smallest_domain_by_degree que smallest_domain. Para n = 10 los resultados también mejoran pero smallest_domain es mejor smallest_domain_by_degree. Para el resto los resultados empeoran como observamos en la tabla.

Heuristicas valores (1 solucion) con <u>FC</u> (elegid BT, FC o RFLA para esta prueba)	n=4	n=6	n=9	n=10	n=11	n=12?
bottom_first	7	18	20	47	28	104
top_first	7	18	20	47	28	104
mid_first	4	15	12	17	13	14

En esta tabla hemos escogido forward-checking (FC) como algoritmo de búsqueda para realizar las ejecuciones, nos hemos basado en obtener una solución y hemos variado los parámetros para heurísticas de valores:

- bottom_first: Instanciación de valores por orden de más pequeño.
- top_first: Instanciación de valores por orden de más grande.
- mid_first: Instanciación de valores por orden de en medio.

Observamos que tanto con bottom_first y top_first obtenemos los mismos resultados que en la primera tabla que hemos analizado. La mejora la obtenemos con mid_first donde observamos una mejora en todas las instanciaciones de n, la mejora es bastante considerable en n = 10 que pasamos de 47 instancias a 17 y de n = 12 que pasamos de 104 instancias a 14.

Parte 2: Sudoku

Modelado del problema

El sudoku implementado es de 9x9 por lo tanto he creado 81 variables enteras que tienen un dominio de 1..9 que son los valores que puede tomar cada celda del sudoku y que representan una matriz de 9 filas y 9 columnas. Las variables tienen una nomenclatura basadas en el patrón ZXY donde X es el número de la fila e Y es el número de la columna del sudoku.

```
# Variables
\vi :
Z11,Z12,Z13,Z14,Z15,Z16,Z17,Z18,Z19,
Z21,Z22,Z23,Z24,Z25,Z26,Z27,Z28,Z29,
Z31,Z32,Z33,Z34,Z35,Z36,Z37,Z38,Z39,
Z41,Z42,Z43,Z44,Z45,Z46,Z47,Z48,Z49,
Z51,Z52,Z53,Z54,Z55,Z56,Z57,Z58,Z59,
Z61,Z62,Z63,Z64,Z65,Z66,Z67,Z68,Z69,
Z71,Z72,Z73,Z74,Z75,Z76,Z77,Z78,Z79,
Z81,Z82,Z83,Z84,Z85,Z86,Z87,Z88,Z89,
Z91,Z92,Z93,Z94,Z95,Z96,Z97,Z98,Z99

# Dominio
1..9 ;
```

Las restricciones las he dividido en cuatro bloques que corresponden a:

- Restricciones para los valores iniciales del sudoku.
- Restricciones por filas.
- Restricciones por columnas.
- Restricciones por bloques.

A continuación vamos a coger cada bloque y vamos a comentar como se ha implementado.

Restricciones para los valores iniciales del sudoku

Este conjunto de restricciones son intensionales y nos permiten asignar valores a las celdas del sudoku que inicialmente tienen un valor, el sudoku implementado es el que nos proporciona la práctica. La nomenclatura de estas restricciones es rvX donde X es el numero de la restricción de variable.

	6		1		4		5	
		8	3		5	6		
2								1
8			4		7			6
		6				3		
7			9		1			4
5								2
		7	2		6	9		
	4		5		8		7	

```
# Restricciones unarias (Declaracion de variables iniciales)
\ci: rv1 , Z12 = 6 ;
\ci: rv2 , Z14 = 1 ;
\ci: rv3 , Z16 = 4 ;
\ci: rv4 , Z18 = 5 ;
\ci: rv5 , Z23 = 8 ;
\ci: rv6 , Z24 = 3 ;
\ci: rv7 , Z26 = 5 ;
\ci: rv8 , Z27 = 6 ;
\ci: rv9 , Z31 = 2 ;
\ci: rv10 , Z39 = 1 ;
\ci: rv11 , Z41 = 8 ;
\ci: rv12 , Z44 = 4 ;
\ci: rv13 , Z46 = 7 ;
\ci: rv14 , Z49 = 6 ;
\ci: rv15 , Z53 = 6 ;
\ci: rv16 , Z57 = 3 ;
\ci: rv17 , Z61 = 7 ;
\ci: rv18 , Z64 = 9 ;
\ci: rv19 , Z66 = 1 ;
\ci: rv20 , Z69 = 4 ;
\ci: rv21 , Z71 = 5 ;
\ci: rv22 , Z79 = 2 ;
\ci: rv23 , Z83 = 7 ;
\ci: rv24 , Z84 = 2 ;
\ci: rv25 , Z86 = 6 ;
\ci: rv26 , Z87 = 9 ;
\ci: rv27 , Z92 = 4 ;
\ci: rv28 , Z94 = 5 ;
\ci: rv29 , Z96 = 8 ;
\ci: rv30 , Z98 = 7 ;
```

Restricciones por filas

En un sudoku se tiene que cumplir que en cada una de sus filas no hayan valores repetidos, esto lo hemos implementado con restricciones intensionales múltiples. La nomenclatura de estas restricciones es rfX donde X es el número de la fila.

```
# Restricciones por filas
\cim : rf1 , <> (Z11,Z12,Z13,Z14,Z15,Z16,Z17,Z18,Z19) ; # Restriccion fila 1
\cim : rf2 , <> (Z21,Z22,Z23,Z24,Z25,Z26,Z27,Z28,Z29) ; # Restriccion fila 2
\cim : rf3 , <> (Z31,Z32,Z33,Z34,Z35,Z36,Z37,Z38,Z39) ; # Restriccion fila 3
\cim : rf4 , <> (Z41,Z42,Z43,Z44,Z45,Z46,Z47,Z48,Z49) ; # Restriccion fila 4
\cim : rf5 , <> (Z51,Z52,Z53,Z54,Z55,Z56,Z57,Z58,Z59) ; # Restriccion fila 5
\cim : rf6 , <> (Z61,Z62,Z63,Z64,Z65,Z66,Z67,Z68,Z69) ; # Restriccion fila 6
\cim : rf7 , <> (Z71,Z72,Z73,Z74,Z75,Z76,Z77,Z78,Z79) ; # Restriccion fila 7
\cim : rf8 , <> (Z81,Z82,Z83,Z84,Z85,Z86,Z87,Z88,Z89) ; # Restriccion fila 8
\cim : rf9 , <> (Z91,Z92,Z93,Z94,Z95,Z96,Z97,Z98,Z99) ; # Restriccion fila 9
```

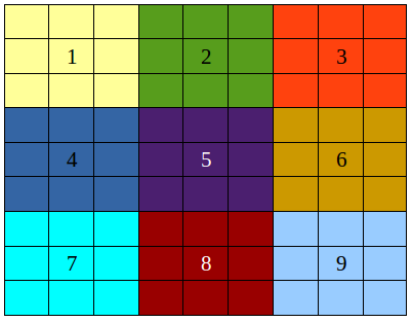
Restricciones por columnas

En un sudoku se tiene que cumplir que en cada una de sus columnas no hayan valores repetidos, esto lo hemos implementado con restricciones intensionales múltiples. La nomenclatura de estas restricciones es rcX donde X es el número de la columna.

```
# Restricciones por columnas
\cim : rc1 , <> (Z11,Z21,Z31,Z41,Z51,Z61,Z71,Z81,Z91) ; # Restriccion col 1
\cim : rc2 , <> (Z12,Z22,Z32,Z42,Z52,Z62,Z72,Z82,Z92) ; # Restriccion col 2
\cim : rc3 , <> (Z13,Z23,Z33,Z43,Z53,Z63,Z73,Z83,Z93) ; # Restriccion col 3
\cim : rc4 , <> (Z14,Z24,Z34,Z44,Z54,Z64,Z74,Z84,Z94) ; # Restriccion col 4
\cim : rc5 , <> (Z15,Z25,Z35,Z45,Z55,Z65,Z75,Z85,Z95) ; # Restriccion col 5
\cim : rc6 , <> (Z16,Z26,Z36,Z46,Z56,Z66,Z76,Z86,Z96) ; # Restriccion col 6
\cim : rc7 , <> (Z17,Z27,Z37,Z47,Z57,Z67,Z77,Z87,Z97) ; # Restriccion col 7
\cim : rc8 , <> (Z18,Z28,Z38,Z48,Z58,Z68,Z78,Z88,Z98) ; # Restriccion col 8
\cim : rc9 , <> (Z19,Z29,Z39,Z49,Z59,Z69,Z79,Z89,Z99) ; # Restriccion col 9
```

Restricciones por bloques

En un sudoku se tiene que cumplir que en cada uno de sus bloques no hayan valores repetidos, esto lo hemos implementado con restricciones intensionales múltiples. La nomenclatura de estas restricciones es rbX donde X es el número de bloque. Un bloque esta formado por 9 celdas y cada bloque lo hemos enumerado de la siguiente manera:




```
# Restricciones por bloques
# Nota: Numeracion de bloques =
#                              1 2 3
#                              4 5 6
#                              7 8 9
\cim : rb1 , <> (Z11,Z12,Z13,Z21,Z22,Z23,Z31,Z32,Z33) ; # Restriccion bloque 1
\cim : rb2 , <> (Z14,Z15,Z16,Z24,Z25,Z26,Z34,Z35,Z36) ; # Restriccion bloque 2
\cim : rb3 , <> (Z17,Z18,Z19,Z27,Z28,Z29,Z37,Z38,Z39) ; # Restriccion bloque 3
\cim : rb4 , <> (Z41,Z42,Z43,Z51,Z52,Z53,Z61,Z62,Z63) ; # Restriccion bloque 4
\cim : rb5 , <> (Z44,Z45,Z46,Z54,Z55,Z56,Z64,Z65,Z66) ; # Restriccion bloque 5
\cim : rb6 , <> (Z47,Z48,Z49,Z57,Z58,Z59,Z67,Z68,Z69) ; # Restriccion bloque 6
\cim : rb7 , <> (Z71,Z72,Z73,Z81,Z82,Z83,Z91,Z92,Z93) ; # Restriccion bloque 7
\cim : rb8 , <> (Z74,Z75,Z76,Z84,Z85,Z86,Z94,Z95,Z96) ; # Restriccion bloque 8
\cim : rb9 , <> (Z77,Z78,Z79,Z87,Z88,Z89,Z97,Z98,Z99) ; # Restriccion bloque 9
```

Una solución mas gráfica

Conflex en nuestro problema nos devuelve la solución desordenada, es decir, nos devuelve las variables desordenadas ya que nos devuelve primero las variables que hemos asignado nosotros en las restricciones para los valores iniciales del sudoku y además es difícil interpretar la solución sin ver una matriz ordenada, por estos motivos he implementado un programa en python al cual le pasas un fichero con la solución que devuelve Conflex y nos pinta la matriz que representa el sudoku y nos comprueba si la solución del sudoku es correcta o si es incorrecta, en el caso de que sea incorrecta nos indica el número de filas, columnas y bloques en los cuales hay un error en la solución.

Primero hemos de generar un fichero con la salida de la ejecución de Conflex:

conflex.exe sudoku.csp > salida_sudoku.txt

Una vez generado el fichero solo hemos de ejecutar el programa de python:

python2.7 sudoku.py salida_sudoku.txt

La salida que nos devuelve el programa python la vemos a continuación:

```
$ python2.7 sudoku.py salida_sudoku.txt
[9, 6, 3, 1, 7, 4, 2, 5, 8]
[1, 7, 8, 3, 2, 5, 6, 4, 9]
[2, 5, 4, 6, 8, 9, 7, 3, 1]
[8, 2, 1, 4, 3, 7, 5, 9, 6]
[4, 9, 6, 8, 5, 2, 3, 1, 7]
[7, 3, 5, 9, 6, 1, 8, 2, 4]
[5, 8, 9, 7, 1, 3, 4, 6, 2]
[3, 1, 7, 2, 4, 6, 9, 8, 5]
[6, 4, 2, 5, 9, 8, 1, 7, 3]


----------------------------------------------------------------------
Solucion correcta!!!
----------------------------------------------------------------------
```

Si por ejemplo editamos el fichero salida_sudoku.txt y modificamos por ejemplo la variable Z17 = 2 por Z17 = 6 observamos que nos devuelve tres errores, error en la fila 1, error en la columna 7 y error en el bloque 3:

```
$ python2.7 sudoku.py salida_sudoku.txt
[9, 6, 3, 1, 7, 4, 6, 5, 8]
[1, 7, 8, 3, 2, 5, 6, 4, 9]
[2, 5, 4, 6, 8, 9, 7, 3, 1]
[8, 2, 1, 4, 3, 7, 5, 9, 6]
[4, 9, 6, 8, 5, 2, 3, 1, 7]
[7, 3, 5, 9, 6, 1, 8, 2, 4]
[5, 8, 9, 7, 1, 3, 4, 6, 2]
[3, 1, 7, 2, 4, 6, 9, 8, 5]
[6, 4, 2, 5, 9, 8, 1, 7, 3]


----------------------------------------------------------------------
Error en la fila:  1
Error en la columna:  7
Error en el bloque:  3
----------------------------------------------------------------------
```

Este programa aparte de proporcionar la solución de forma gráfica nos ha servido para verificar que nuestro csp devuelve la solución correcta ya que como hemos visto en las restricciones es fácil cometer un error humano introduciéndolas ya que trabajamos con 81 variables.

Resultados obtenidos

En este apartado vamos a realizar una evaluación de nuestro modelo sudoku.csp en función de algoritmos de búsqueda, heurísticas sobre variables y heurísticas sobre valores.

Como ya sabemos un sudoku solo tiene una solución por lo tanto nuestro csp solo devuelve una solución y por ello solo trabajamos con una solución.

Algoritmo de búsqueda	N.º de instanciaciones
BT	3561
FC	85
RFLA	82

Heurísticas sobre variables (FC)	N.º de instanciaciones
Static: smallest_domain	85
Static: smallest_domain_by_degree	85
Dynamic: smallest_domain	83
Dynamic: smallest_domain_by_deg	81

Heurísticas sobre valores (FC)	N.º de instanciaciones
bottom_first	82
top_first	82
mid_first	82

Observamos que los mejores parámetros para resolver el CSP con conflex respecto al estudio que hemos realizado son:

- Algoritmo de búsqueda: FC
- Heurística sobre variables: smallest_domain_by_deg
- Tipo de heurística sobre variables: Dinámica

Con un total de 81 instancias generadas para resolver nuestro problema.

La peor es mediante backtracking (BT) que como ya sabemos es la menos informada y necesita 3561 instancias para encontrar la solución.

Sobre la tabla de heurísticas sobre variables observamos que la heurística dinámica es mejor que la heurística estática y sobre las heurísticas sobre valores observamos que para los tres métodos el número de instanciaciones es el mismo, 82.

Parte 3: Las reuniones

Modelado del problema

Nuestro problema esta compuesto de 6 amigos y sus respectivas esposas los cuales desean asistir a dos reuniones que se celebran en un hotel. Tenemos 12 variables enteras en nuestro modelo las cuales se dividen a su vez en dos grupos, un grupo hace referencia a los amigos cuya nomenclatura es MX donde M hace referencia a marido y X es la inicial del nombre de cada amigo (ya que ningún nombre de los amigos comienza por la misma letra) por lo tanto tenemos:

MA = Marido Antonio; MD = Marido David; ML = Marido Luis; MP = Marido Pedro;

ME = Marido Enrique; MR = Marido Ramón

La nomenclatura para las esposas de dichos maridos es EX donde E hace referencia a esposa_de y la X es la inicial del marido de dicha esposa, por lo tanto tenemos:

EA = Esposa_de Antonio; ED = Esposa_de David; EL = Esposa_de Luis; EP = Esposa_de Pedro;

EE = Esposa_de Enrique; ER = Esposa_de Ramón

El dominio para las 12 variables (tanto para maridos como para esposas) es 0..1 que hace referencia a la reunión (0 o 1) a la cual va a asistir cada persona.

```
\vi :
MA,MD,ML,MP,ME,MR, # Maridos
EA,ED,EL,EP,EE,ER  # Esposas

# Dominio (Reunion 0 o 1)
0..1 ;
```

El problema nos define nueve restricciones que se han de cumplir, de las cuales de la 1 a la 7 son restricciones condicionales, la 8 es una disyunción de restricciones formada por dos restricciones condicionales y la 9 es una disyunción de restricciones con 6 restricciones intensionales. Las restricciones siguen la nomenclatura RX donde X es el número de la restricción. A continuación vamos indicar que significa cada restricción y como se ha implementado en nuestro modelo reuniones.csp.

R1: if(MA=EA && MD=ED && ML=EP) entonces ME=ER

```
\cc: R1
    \if     \ci: r1a, MA = EA;
            \ci: r1b, MD = ED;
            \ci: r1c, ML = EP;
    \then   \ci: r1d, ME = ER;
;
```

R2: if(MA=EA && MP=EP && MD=EE) entonces MR!=EL

```
\cc: R2
    \if     \ci: r2a, MA = EA;
            \ci: r2b, MP = EP;
            \ci: r2c, MD = EE;
    \then   \ci: r2d, MR != EL;
;
```

R3: if(ME=MR && MR=EE && EE=ER && MA!=ED) entonces ML!=EP

```
\cc: R3
    \if     \ci: r3a, ME = MR;
            \ci: r3b, MR = EE;
            \ci: r3c, EE = ER;
            \ci: r3d, MA != ED;
    \then   \ci: r3e, ML != EP;
;
```

R4: if(MA=EA && MR=ER && MD!=EE) entonces ML=EP

```
\cc: R4
    \if     \ci: r4a, MA = EA;
            \ci: r4b, MR = ER;
            \ci: r4c, MD != EE;
    \then   \ci: r4d, ML = EP;
;
```

R5: if(ML=EL && MP=EP && ME=ER) entonces MA!=ED

```
\cc: R5
    \if     \ci: r5a, ML = EL;
            \ci: r5b, MP = EP;
            \ci: r5c, ME = ER;
    \then   \ci: r5d, MA != ED;
;
```

R6: if(MD=ME && ME=ED && ED=EE && ML!=EP) entonces MR=EL

```
\cc: R6
    \if     \ci: r6a, MD = ME;
            \ci: r6b, ME = ED;
            \ci: r6c, ED = EE;
            \ci: r6d, ML != EP;
    \then   \ci: r6e, MR = EL;
;
```

R7: if(MP=EA) entonces MA=EA

```
\cc: R7
    \if     \ci: r7a, MP = EA;
    \then   \ci: r7b, MA = EA;
;
```

R8: if(EP=EL) entonces (ML=EL o MP=EP)

```
\cc: R8
    \if     \ci: r7a, EP = EL;
    \then   \ci: r7b, ML != MP;
;
```

R9: Hay al menos un matrimonio cuyos miembros no están juntos en la misma reunión.

```
\doc: R9
    \ci: r9a, MA != EA;
\or
    \ci: r9b, MD != ED;
\or
    \ci: r9c, ML != EL;
\or
    \ci: r9d, MP != EP;
\or
    \ci: r9e, ME != EE;
\or
    \ci: r9f, MR != ER;
;
```

R10: Opcional, En ninguna reunión puede haber menos de 4 personas del grupo de los seis amigos y sus esposas.

```
\ci: R10a, MA + MD + ML + MP + ME + MR + EA + ED + EL + EP + EE + ER > 4;

\ci: R10b, MA + MD + ML + MP + ME + MR + EA + ED + EL + EP + EE + ER < 9;
```

Resultados obtenidos

En este apartado vamos a realizar una evaluación de nuestro modelo reuniones.csp en función de algoritmos de búsqueda, heurísticas sobre variables y heurísticas sobre valores.

Algoritmo de búsqueda (1 solución)	N.º de instanciaciones
BT	35
FC	12
RFLA	19

Algoritmo de búsqueda (Todas soluciones)	N.º de instanciaciones
BT	2942
FC	522
RFLA	878

Heurísticas sobre variables (FC)	N.º de instanciaciones
Static: smallest_domain	12
Static: smallest_domain_by_degree	2*
Dynamic: smallest_domain	10*
Dynamic: smallest_domain_by_deg	2*

Heurísticas sobre valores (FC)	N.º de instanciaciones
bottom_first	12
top_first	14
mid_first	14

En la primera tabla hemos realizado ejecuciones para obtener una solución variando el algoritmo de búsqueda, observamos como es lógico que BT es el que mas instanciaciones genera (es la menos informada) la que menos instancias proporciona es FC con 12.

En la segunda tabla hemos realizado ejecuciones para obtener todas las soluciones variando el algoritmo de búsqueda, después de realizar varias comprobaciones a mi CSP y verificar que esta bien formulado he llegado a la conclusión de que conflex tiene un error o no gestiona correctamente la ramificación y poda en sus algoritmos de búsqueda ya que variando el algoritmo de búsqueda nos devuelve diferente número de soluciones y debería devolver el mismo número para todos los algoritmos ya que son todas las soluciones que existen para nuestro problema, los resultados que me

ha devuelto según número total de soluciones los muestro a continuación:

- BT: 800 soluciones.
- FC: 97 soluciones.
- RFLA: 165 soluciones.

En la tercera tabla hemos trabajado con una solución, con heurísticas sobre variables (estáticas y dinámicas) y con el algoritmo de búsqueda FC, los valores marcados con un * en la tabla hacen referencia a que con esa heurística no se ha encontrado ninguna solución.
Respecto a la última tabla hemos trabajado con una solución, con heurísticas sobre valores y con el algoritmo de búsqueda FC, con bottom_first hemos obtenido 12 instancias, con las otras dos hemos obtenido 14 instancias.
Este apartado me ha llevado a la conclusión de que como he comentado antes conflex tiene algún error que supongo tiene que ver con restricciones de tipo condición (if) ya que cuando le pedimos que nos devuelva todas las soluciones y empleamos para ello diferentes métodos de búsqueda el número de soluciones que devuelve es diferente. Para demostrar que mi CSP es correcto he implementado en python un programa (reuniones.py) que comprueba solución a solución si cumple con las restricciones establecidas en el modelo, y lo demuestro a continuación:

- Con el fichero BT que contiene 800 soluciones:

```
$ python2.7 reuniones.py salida_reuniones/reuniones_bt.txt
Todas las soluciones analizadas son correctas
```

- Con el fichero FC que contiene 97 soluciones:

```
$ python2.7 reuniones.py salida_reuniones/reuniones_fc.txt
Todas las soluciones analizadas son correctas
```

- Con el fichero RFLA que contiene 165 soluciones:

```
$ python2.7 reuniones.py salida_reuniones/reuniones_rfla.txt
Todas las soluciones analizadas son correctas
```

Todos las salidas *.txt las adjunto en la entrega, pero las puedes obtener fácilmente redireccionando la salida de conflex de la siguiente manera:

conflex.exe reuniones.csp > salida_reuniones/reuniones_xxx.txt

Conclusión

En esta práctica hemos visto como resolver problemas de satisfacción de restricciones, resolviendo problemas reales como son un sudoku y el problema de las reuniones, por otra parte hemos visto que no todos los problemas se pueden resolver con un CSP estándar y por lo tanto necesitamos CSP's flexibles como hemos estudiado e implementado en el problema de las hamburguesas.
En esta práctica hemos utilizado conflex que es una herramienta para la resolución de CSPs, es libre, con una interfaz simple y que mediante el problema de las n reinas hemos podido comprender como es su sintaxis y como es la salida que nos proporciona con la finalidad de tener a mano una herramienta para cualquier problema CSP que nos pudiéramos encontrar en un futuro.

Autocrítica

Creo que esta práctica me ha ayudado mucho a complementar lo visto en teoría acerca de lo que son los CSPs ya que hemos tenido que modelarlos y utilizar conflex para resolverlos. Respecto al apartado 3 de la práctica he tenido que leer bastante parte del manual que nos proporcionáis en la misma y como he comentado he llegado a la conclusión de que es un error de conflex. Creo que esta práctica ha sido muy completa y muy interesante ya que en otras asignaturas hemos visto de forma completamente analítica modelos como el algoritmo simplex para resolver problemas con restricciones pero no los habíamos planteado basándonos en técnicas computacionales en las cuales observamos la importancia de la IA con conceptos sobre búsqueda de soluciones y heurísticas al enfrentarnos a problemas NP.