Solución eficiente de Sudokus por medio del problema de satisfacción de restricciones o CSP

Autor: Adelina Figueira
Carnet: 15-10484

1. Objetivos planteados y justificación

El objetivo principal de este proyecto es verificar cuáles de los algoritmos utilizados para la resolución del problema de satisfacción de restricciones son capaces de resolver problemas sencillos de manera rápida. El problema seleccionado a resolver es el juego de Sudoku, para diversos tamaños del juego. Se incluyeron los tamaños 9x9, 16x16 y 25x25.

1.1. Objetivos específicos

- Conocer que heurísticas son capaces de retornar el resultado de un Sudoku en el menor tiempo posible.
- Encontrar que algoritmos de propagación de restricciones son más eficientes para la resolución del problema del Sudoku.
- Verificar si el uso de forward-checking durante el backtracking mejora o no el rendimiento de los algoritmos CSP para la resolución del Sudoku.

1.2. Justificación

Estos experimentos nos permitirán apreciar las bondades de cada uno de los algoritmos que existen para el problema de satisfacción de restricciones o CSP utilizando como ejemplo el problema de Sudoku, diversos volúmenes de datos con los que se trabaja y permite observar de manera práctica que algoritmos utilizar en caso de que no se consiga una solución o esta tenga un tiempo de ejecución bastante largo.

2. Técnicas utilizadas

Las técnicas utilizadas en este proyecto son los algoritmos vistos en clase, específicamente los algoritmos relacionados a la resolución del problema de satisfacción de restricciones o CSP. Entre estos algoritmos se encuentran:

- Heurísticas : MRV (Valores mínimos restantes), primera variable no asignada, valores de dominio no ordenados y LCV (Valores con menos restricciones).
- Forward Checking
- MAC (Mantener la consistencia de arco)
- AC3
- AC4
- Backtracking

2.1. Backtracking

El algoritmo de backtracking es uno de los algoritmos de búsqueda que se utilizan para hacer la propagación de las restricción al resolver un problema CSP. Este algoritmo es recursivo y en cada iteración selecciona una variable (basándose en una heurística), luego se busca el dominio de esta variable para asignarle valores dentro de su dominio. Posteriormente se verifica que las restricciones se cumplan y se le van asignando estos valores, únicamente se toma otro valor si no se consigue un valor que cumpla con las restricciones. Al probar todas las asignaciones disponibles para una variable, el algoritmo termina. En caso de que una asignación no cumpla con las restricciones se elimina de los valores disponibles, de esta manera al culminar el algoritmo, solo estarán disponibles los valores que resuelven el problema con las restricciones establecidas. El algoritmo de backtracking utilizado en este proyecto es:

```
def backtracking_search(csp, select_unassigned_variable=first_unassigned_variable,
                        order_domain_values=unordered_domain_values, inference=no_inference):
    def backtrack(assignment):
        if len(assignment) == len(csp.variables):
            return assignment
        var = select_unassigned_variable(assignment, csp)
        for value in order_domain_values(var, assignment, csp):
            if 0 == csp.nconflicts(var, value, assignment):
                csp.assign(var, value, assignment)
                removals = csp.suppose(var, value)
                if inference(csp, var, value, assignment, removals):
                    result = backtrack(assignment)
                    if result is not None:
                        return result
                csp.restore(removals)
        csp.unassign(var, assignment)
        return None

    result = backtrack({})
    assert result is None or csp.goal_test(result)
```

2.2. Forward Checking

El algoritmo forward-checking es una variación del algoritmo de backtracking que disminuye el espacio de búsqueda haciendo uso de un tipo de consistencia local. Para cada variable no asignada, se mantiene una lista de los valores restantes y se aplican las restricciones para eliminar las inconsistencias en los valores de este conjunto. El forward-checking visita los vecinos de una variable, para verificar si alguno de los valores restantes es inconsistente (no cumple con las restricciones) y lo elimina si lo es. El algoritmo se devuelve de manera similar al backtracking en caso de que la variable no tenga más valores. El algoritmo de forward-checking utilizado en el proyecto es:

```
def forward_checking(csp, var, value, assignment, removals):
    """Prune neighbor values inconsistent with var=value."""
    csp.support_pruning()
    for B in csp.neighbors[var]:
        if B not in assignment:
            for b in csp.curr_domains[B][:]:
                if not csp.constraints(var, value, B, b):
                    csp.prune(B, b, removals)
            if not csp.curr_domains[B]:
                return False
    return True
```

2.3. Propagación de restricciones

Las técnicas o algoritmos de propagación de restricciones se utilizan para modificar un CSP. Permiten forzar a la consistencia local y quedará un problema más simple y sencillo de resolver. Para este proyecto se utilizaron 3 de estas técnicas:

2.4. AC3

Es uno de las técnicas de propagación de restricciones más populares que garantiza la consistencia de arco. El algoritmo AC3 se encarga de examinar los arcos entre los pares de variables (x,y). Remueve los valores del dominio de x que no son consistentes con las restricciones entre x e y. Se mantienen alojados en un conjunto los arcos que faltan por verificar, cuando al dominio de una variable se le ha removido algún valor. Todos los arcos de una restricción apuntan a la variable que ha sido podada (excepto el arco de la restricción actual) y se agregan a este conjunto. Este algoritmo puede terminar porque las variables son finitas y al menos 1 arco o 1 valor se remueve en cada paso del algoritmo. El algoritmo AC3 utilizado en el proyecto es:

```
def AC3(csp, queue=None, removals=None, arc_heuristic=dom_j_up):
    """[Figure 6.3]"""
    if queue is None:
```

```
        queue = {(Xi, Xk) for Xi in csp.variables for Xk in csp.neighbors[Xi]}
    csp.support_pruning()
    queue = arc_heuristic(csp, queue)
    checks = 0
    while queue:
        (Xi, Xj) = queue.pop()
        revised, checks = revise(csp, Xi, Xj, removals, checks)
        if revised:
            if not csp.curr_domains[Xi]:
                return False, checks  # CSP is inconsistent
            for Xk in csp.neighbors[Xi]:
                if Xk != Xj:
                    queue.add((Xk, Xi))
    return True, checks  # CSP is satisfiable


def revise(csp, Xi, Xj, removals, checks=0):
    """Return true if we remove a value."""
    revised = False
    for x in csp.curr_domains[Xi][:]:
        # If Xi=x conflicts with Xj=y for every possible y, eliminate Xi=x
        # if all(not csp.constraints(Xi, x, Xj, y) for y in csp.curr_domains[Xj]):
        conflict = True
        for y in csp.curr_domains[Xj]:
            if csp.constraints(Xi, x, Xj, y):
                conflict = False
            checks += 1
            if not conflict:
                break
        if conflict:
            csp.prune(Xi, x, removals)
            revised = True
    return revised, checks
```

2.5. AC4

Es un algoritmo de consistencia de arco que se creó para mejorar el algoritmo AC3. AC4 es considerado el mejor algoritmo de consistencia de arco para CSP en muchos casos. El algoritmo AC4 está basado en la siguiente definición:
Dado un valor a para la variable X_i, decimos que a es soportada por X_j si existe al menos un valor b en $Dom(X_j)$ tal que $X_i = a$ y $X_j = b$ son compatibles.
Los valores que no son soportados, son redundantes y se eliminan. Este algoritmo consiste de dos partes: Un conjunto de arcos y contadores. Los valores redundantes son removidos del dominio de la variable, y durante este proceso se actualizan los contadores que puede generar nuevos valores redundantes que han sido removidos anteriormente. En el contador para cada arco hay un número que registra el número de soportes que recibe de cada restricción que incluya la variable, si este contador llega a 0, se elimina del dominio de la variable. El conjunto de soporte para el valor de cada variable contiene todas las variables que lo soportan. Aunque el algoritmo AC4 tiene una mejor complejidad que AC3, se requiere de un gran tamaño para alojar los conjuntos, por lo que puede tener un peor desempeño al tener grandes cantidades de datos. El algoritmo AC4 utilizado en el proyecto es:

```
def AC4(csp, queue=None, removals=None, arc_heuristic=dom_j_up):
    if queue is None:
        queue = {(Xi, Xk) for Xi in csp.variables for Xk in csp.neighbors[Xi]}
    csp.support_pruning()
    queue = arc_heuristic(csp, queue)
    support_counter = Counter()
    variable_value_pairs_supported = defaultdict(set)
    unsupported_variable_value_pairs = []
    checks = 0
    # construction and initialization of support sets
```

```
    while queue:
        (Xi, Xj) = queue.pop()
        revised = False
        for x in csp.curr_domains[Xi][:]:
            for y in csp.curr_domains[Xj]:
                if csp.constraints(Xi, x, Xj, y):
                    support_counter[(Xi, x, Xj)] += 1
                    variable_value_pairs_supported[(Xj, y)].add((Xi, x))
                checks += 1
            if support_counter[(Xi, x, Xj)] == 0:
                csp.prune(Xi, x, removals)
                revised = True
                unsupported_variable_value_pairs.append((Xi, x))
        if revised:
            if not csp.curr_domains[Xi]:
                return False, checks  # CSP is inconsistent
    # propagation of removed values
    while unsupported_variable_value_pairs:
        Xj, y = unsupported_variable_value_pairs.pop()
        for Xi, x in variable_value_pairs_supported[(Xj, y)]:
            revised = False
            if x in csp.curr_domains[Xi][:]:
                support_counter[(Xi, x, Xj)] -= 1
                if support_counter[(Xi, x, Xj)] == 0:
                    csp.prune(Xi, x, removals)
                    revised = True
                    unsupported_variable_value_pairs.append((Xi, x))
            if revised:
                if not csp.curr_domains[Xi]:
                    return False, checks  # CSP is inconsistent
    return True, checks  # CSP is satisfiable
```

2.6. MAC

El algoritmo MAC o Mantaining Arc Consistency desarrollado en este proyecto, se trata del algoritmo AC3 con doble soporte en la heurística de dominio. La librería AIMA contiene una versión mejorada del algoritmo AC3 nombrada AC3b que utiliza como heurística un conjunto de los valores restantes consistentes de una variable. Con esta heurística se rellena la cola queue que se inicializa al comenzar el algoritmo. La heurística que se utiliza para este algoritmo es la heurística Domjup que ordena por tamaño los dominios para cada uno de los arcos que forman parte del problema. El algoritmo MAC utilizado en el proyecto es:

```
def mac(csp, var, value, assignment, removals, constraint_propagation=AC3b):
    """Maintain arc consistency."""
    return constraint_propagation(csp, {(X, var) for X in csp.neighbors[var]}, removals)

def AC3b(csp, queue=None, removals=None, arc_heuristic=dom_j_up):
    if queue is None:
        queue = {(Xi, Xk) for Xi in csp.variables for Xk in csp.neighbors[Xi]}
    csp.support_pruning()
    queue = arc_heuristic(csp, queue)
    checks = 0
    while queue:
        (Xi, Xj) = queue.pop()
        # Si_p values are all known to be supported by Xj
        # Sj_p values are all known to be supported by Xi
        # Dj - Sj_p = Sj_u values are unknown, as yet, to be supported by Xi
        Si_p, Sj_p, Sj_u, checks = partition(csp, Xi, Xj, checks)
        if not Si_p:
            return False, checks  # CSP is inconsistent
        revised = False
        for x in set(csp.curr_domains[Xi]) - Si_p:
```

```
                csp.prune(Xi, x, removals)
                revised = True
            if revised:
                for Xk in csp.neighbors[Xi]:
                    if Xk != Xj:
                        queue.add((Xk, Xi))
            if (Xj, Xi) in queue:
                if isinstance(queue, set):
                    # or queue -= {(Xj, Xi)} or queue.remove((Xj, Xi))
                    queue.difference_update({(Xj, Xi)})
                else:
                    queue.difference_update((Xj, Xi))
                # the elements in D_j which are supported by Xi are given by the union of Sj_p with the set
                # elements of Sj_u which further processing will show to be supported by some vi_p in Si_p
                for vj_p in Sj_u:
                    for vi_p in Si_p:
                        conflict = True
                        if csp.constraints(Xj, vj_p, Xi, vi_p):
                            conflict = False
                            Sj_p.add(vj_p)
                        checks += 1
                        if not conflict:
                            break
                revised = False
                for x in set(csp.curr_domains[Xj]) - Sj_p:
                    csp.prune(Xj, x, removals)
                    revised = True
                if revised:
                    for Xk in csp.neighbors[Xj]:
                        if Xk != Xi:
                            queue.add((Xk, Xj))
    return True, checks  # CSP is satisfiable

def dom_j_up(csp, queue):
    return SortedSet(queue, key=lambda t: neg(len(csp.curr_domains[t[1]])))
```

2.7. Heurísticas

2.8. MRV

Esta heurística es una de las heurísticas utilizadas para seleccionar la variable no asignada al hacer el backtracking. Consiste de tomar el valor mínimo restante para una asignación que está en las variables del problema y no incumple las restricciones establecidas. El algoritmo de MRV utilizado en el proyecto es:

```
def mrv(assignment, csp):
    """Minimum-remaining-values heuristic."""
    return argmin_random_tie([v for v in csp.variables if v not in assignment],
                             key=lambda var: num_legal_values(csp, var, assignment))
```

2.9. Primera variable no asignada

Esta heurística es una de las heurísticas utilizadas para seleccionar la variable no asignada al hacer el backtracking. A diferencia del MRV, solo toma la primera variable que se consiga y no esté en los valores ya asignados. El algoritmo utilizado en el proyecto es:

```
def first_unassigned_variable(assignment, csp):
    """The default variable order."""
    return first([var for var in csp.variables if var not in assignment])
```

2.10. LCV

Esta heurística es una de las heurísticas utilizadas para seleccionar el orden de los valores del dominio de una variable. Selecciona los valores que no han sido verificados para una variable en particular y retorna estos

valores ordenados. El algoritmo de LCV utilizado en el proyecto es:

```
def lcv(var, assignment, csp):
    """Least-constraining-values heuristic."""
    return sorted(csp.choices(var), key=lambda val: csp.nconflicts(var, val, assignment))

def nconflicts(self, var, val, assignment):
    """The number of conflicts, as recorded with each assignment.
    Count conflicts in row and in up, down diagonals. If there
    is a queen there, it can't conflict with itself, so subtract 3."""
    n = len(self.variables)
    c = self.rows[val] + self.downs[var + val] + self.ups[var - val + n - 1]
    if assignment.get(var, None) == val:
        c -= 3
    return c
```

2.11. Valores del dominio no ordenados

Esta heurística es una de las heurísticas utilizadas para seleccionar el orden de los valores del dominio de una variable. A diferencia de la heurística anterior, esta heurísitca únicamente retorna los valores del dominio de una variable sin verificar si se trata o no de valores ya tomados y tampoco son ordenados. El algoritmo utilizado en el proyecto es:

```
def unordered_domain_values(var, assignment, csp):
    """The default value order."""
    return csp.choices(var)
```

3. Antecedentes

En su trabajo 'Why AC-3 is Almost Always Better than AC-4 for Establishing Arc Consistency in CSPs', Richard Wallace concluye que a pesar que existen algunos problemas para los que AC3 es menos eficiente que AC4, este algoritmo es mejor que AC4 para CSPs basados en relaciones generalizadas. Los problemas que minimizan la eficiencia de los algoritmos de consistencia de arco, como dominios grandes o un gran número de restricciones, suelen tener un mayor efecto en AC4 que en AC3.
El algoritmo AC3 puede simplificar un problema al reducir el dominio de las variables, basándose en la consistencia de las restricciones. Además puede manejar varios tipos de restricciones, es eficiente y se puede utilizar en conjunto con otros algoritmos para CSPs. Pero algunas de las limitaciones de este algoritmo son su complejidad, en tiempo de cómputo para ciertas redes grandes, que la poda del dominio de las variables sea agresiva y se pierdan datos importantes y este algoritmo asume que las restricciones son binarias, por lo que debe ser modificado al aplicarse a restricciones no binarias.

4. Herramientas y librerías

Se utilizó la librería AIMA desarrollada inicialmente por Peter Norvig. Se utilizaron únicamente las implementaciones de los algoritmos para los problemas de satisfacción de restricciones. La librería incluye un problema para Sudokus de tamaño 9x9 pero este se modificó para que permitiese la entrada de tableros de Sudokus con un tamaño mayor 9x9 y que este valor no sea fijo. La librería AIMA desarrolla el problema de Sudoku con el uso de la restricción que indica que dos cuadros vecinos no deben tener el mismo valor, hacia arriba o hacia abajo.
También se hizo uso de la librería multiprocessing de Python para lograr un timeout al momento que las corridas de los algoritmos tengan un tiempo mayor a 2 minutos, debido a que estos resultados son bastante ineficientes. Si alguna combinación de los algoritmos genera un tiempo mayor a 120 segundos, se termina la ejecución de la función y continúa con los siguientes valores.

5. Descripción de los experimentos realizados

Se realizaron varios experimentos para Sudokus de diferentes tamaños, desde tableros de 9x9, 16x16 hasta 25x25. Se probaron todos los algoritmos mencionados en la sección 2, para los siguientes tableros de Sudoku.

- Tamaño 9x9 Vacío (EMPTY)

- Tamaño 9x9 Fácil (EASY1)
- Tamaño 9x9 Difícil (HARD1)
- Tamaño 9x9 Difícil 2 (HARD2)
- Tamaño 16x16 Vacío (EMPTY)
- Tamaño 16x16 Difícil (HARD16)
- Tamaño 25x25 Vacío (EMPTY)
- Tamaño 25x25 Difícil (HARD25)

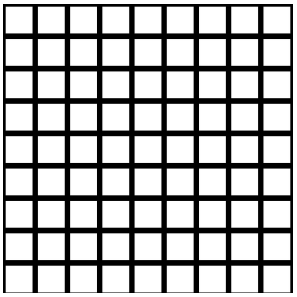
Las tablas a continuación cuentan con un encabezado que define diversos algoritmos y heurísticas utilizadas en la prueba con las siguientes variables:

Algoritmo/Heurística	Variable utilizada
AC3	AC3
AC4	AC4
First Unassigned Variable	FU
MRV	MRV
Unordered domain values	UD
LCV	LCV
No inference	NI
Forward Checking	FC
MAC	MAC

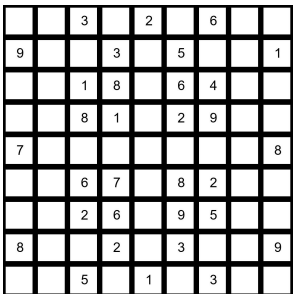
Los tableros utilizados para las pruebas son los siguientes:

5.0.1. Tamaño 9x9

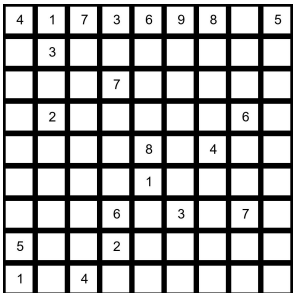
EMPTY



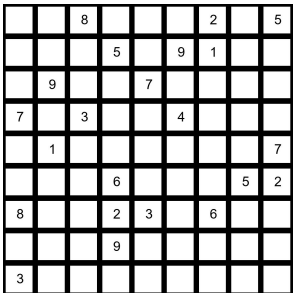
EASY1



HARD1

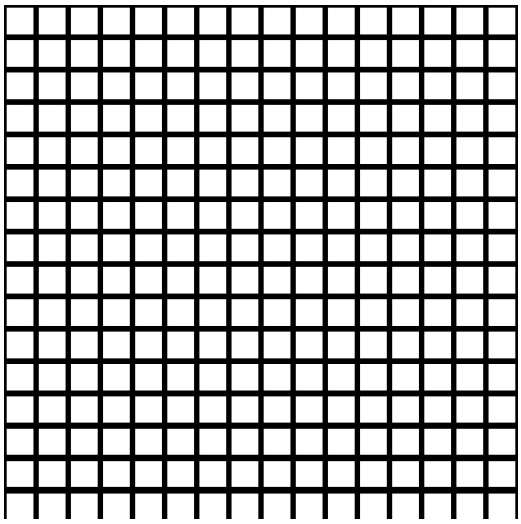


HARD2

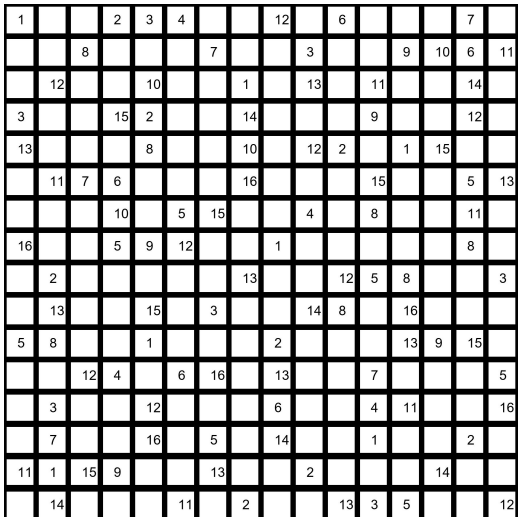


5.0.2. Tamaño 16x16

EMPTY

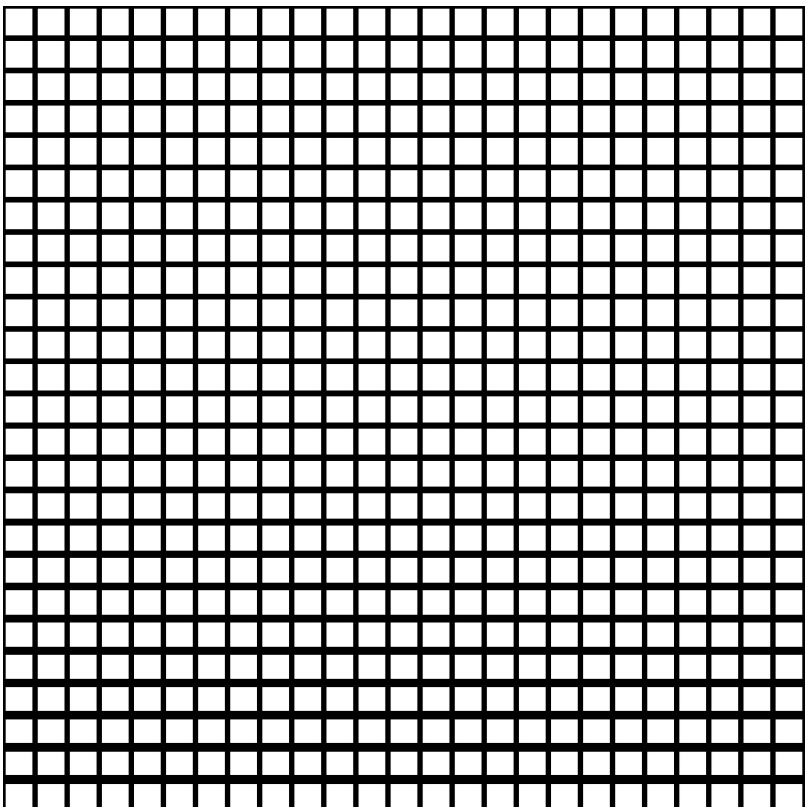


HARD16

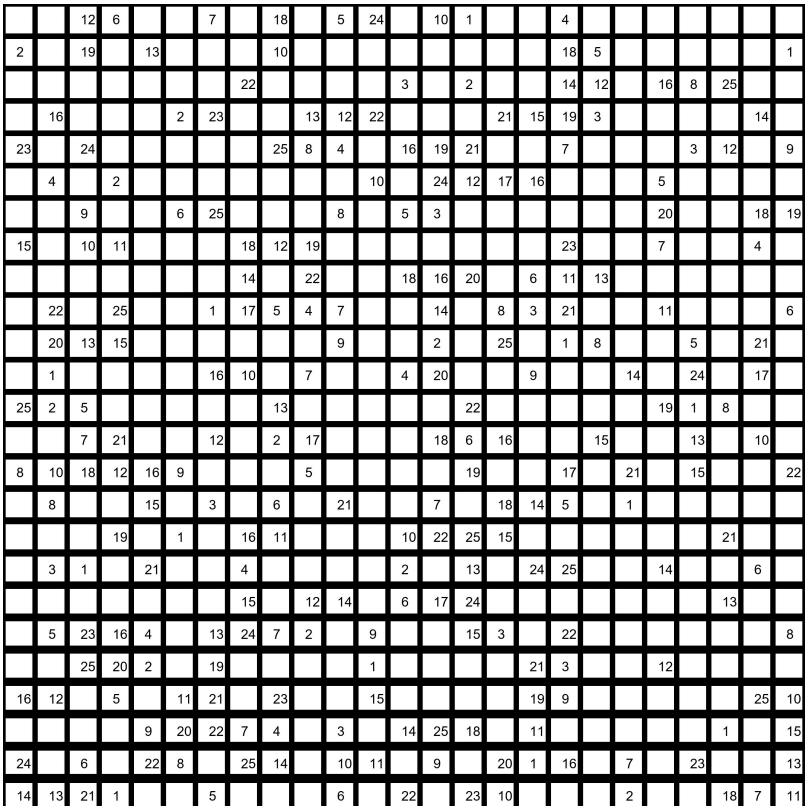


5.0.3. Tamaño 25x25

EMPTY



HARD25



6. Experimentos

6.1. Tablero 9x9

Resultados para tablero vacío

Sudoku	Tiempo	AC3	AC4	FU	MRV	UD	LCV	NI	FC	MAC
EMPTY	0.0161051750183105			X		X		X		
EMPTY	0.0162785053253174			X		X			X	
EMPTY	0			X		X				X
EMPTY	0			X			X	X		
EMPTY	0.0150055885314941			X			X		X	
EMPTY	0			X			X			X
EMPTY	0.0156326293945313				X	X		X		
EMPTY	0				X	X			X	
EMPTY	0.0156264305114746				X	X				X
EMPTY	0				X		X	X		
EMPTY	0				X		X		X	
EMPTY	0.0156266689300537				X		X			X
EMPTY	0	X		X		X		X		
EMPTY	0.0156240463256836	X		X		X			X	
EMPTY	0.0206899642944336	X		X		X				X
EMPTY	0.00816464424133301	X		X			X	X		
EMPTY	0.0164527893066406	X		X			X		X	
EMPTY	0.00841665267944336	X		X			X			X
EMPTY	0.00850486755371094	X			X	X		X		
EMPTY	0	X			X	X			X	
EMPTY	0.015632152557373	X			X	X				X
EMPTY	0	X			X		X	X		
EMPTY	0.0162258148193359	X			X		X		X	
EMPTY	0	X			X		X			X
EMPTY	0.109309673309326		X	X		X		X		
EMPTY	0.0156264305114746		X	X		X			X	
EMPTY	0.0167396068572998		X	X		X				X
EMPTY	0		X	X			X	X		
EMPTY	0.0150060653686523		X	X			X		X	
EMPTY	0.0156326293945313		X	X			X			X
EMPTY	0.0156257152557373		X		X	X		X		
EMPTY	0.0159063339233398		X		X	X			X	
EMPTY	0		X		X	X				X
EMPTY	0.015634298324585		X		X		X	X		
EMPTY	0		X		X		X		X	
EMPTY	0.0156247615814209		X		X		X			X
Promedio:	0.0121969845559862		Mínimo:	0		Máximo:	0.109309673309326			

Resultados para tablero EASY1

Sudoku	Tiempo	AC3	AC4	FU	MRV	UD	LCV	NI	FC	MAC
EASY1	0.0150058269500732			X		X		X		
EASY1	0			X		X			X	
EASY1	0.0193843841552734			X		X				X
EASY1	0			X			X	X		
EASY1	0.012005090713501			X			X		X	
EASY1	0			X			X			X
EASY1	0				X	X		X		
EASY1	0.0156333446502686				X	X			X	
EASY1	0				X	X				X
EASY1	0.0156245231628418				X		X	X		
EASY1	0				X		X		X	
EASY1	0				X		X			X
EASY1	0.0156261920928955	X		X		X		X		
EASY1	0.0156247615814209	X		X		X			X	
EASY1	0.0156242847442627	X		X		X				X
EASY1	0	X		X			X	X		
EASY1	0.0156266689300537	X		X			X		X	
EASY1	0.0156257152557373	X		X			X			X
EASY1	0	X			X	X		X		
EASY1	0.0156228542327881	X			X	X			X	
EASY1	0	X			X	X				X
EASY1	0.0156242847442627	X			X		X	X		
EASY1	0.0156252384185791	X			X		X		X	
EASY1	0	X			X		X			X
EASY1	0.0156261920928955		X	X		X		X		
EASY1	0.0156245231628418		X	X		X			X	
EASY1	0.0156257152557373		X	X		X				X
EASY1	0.0156245231628418		X	X			X	X		
EASY1	0		X	X			X		X	
EASY1	0.0156245231628418		X	X			X			X
EASY1	0.0156261920928955		X		X	X		X		
EASY1	0		X		X	X			X	
EASY1	0.0156238079071045		X		X	X				X
EASY1	0		X		X		X	X		
EASY1	0.015625		X		X		X		X	
EASY1	0.0159809589385986		X		X		X			X
Promedio:	0.00954540570576986		Mínimo:	0		Máximo:	0.0193843841552734			

Resultados para tablero HARD1

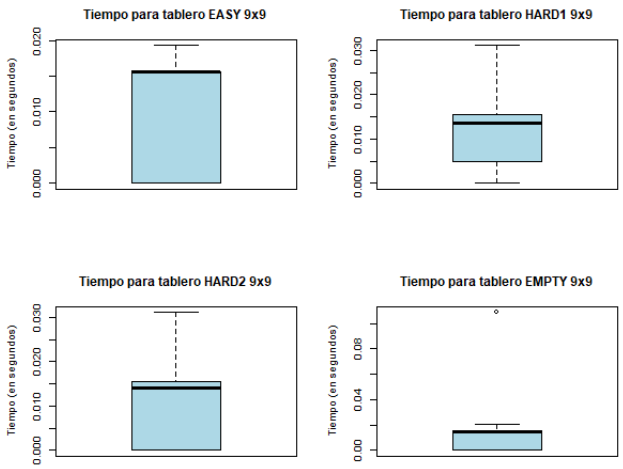
Sudoku	Tiempo	AC3	AC4	FU	MRV	UD	LCV	NI	FC	MAC
HARD1	0			X		X		X		
HARD1	0.0156261920928955			X		X			X	
HARD1	0.0156257152557373			X		X				X
HARD1	0			X			X	X		
HARD1	0.0158312320709229			X			X		X	
HARD1	0.00494003295898438			X			X			X
HARD1	0.00696969032287598				X	X		X		
HARD1	0.0066072940826416				X	X			X	
HARD1	0				X	X				X
HARD1	0.013512134552002				X		X	X		
HARD1	0.00706315040588379				X		X		X	
HARD1	0.00864648818969727				X		X			X
HARD1	0.0187790393829346	X		X		X		X		
HARD1	0.01357102394104	X		X		X			X	
HARD1	0.00707006454467773	X		X		X				X
HARD1	0.0137908458709717	X		X			X	X		
HARD1	0.0135123729705811	X		X			X		X	
HARD1	0.0139431953430176	X		X			X			X
HARD1	0.00681805610656738	X			X	X		X		
HARD1	0.00858712196350098	X			X	X			X	
HARD1	0	X			X	X				X
HARD1	0.0155048370361328	X			X		X	X		
HARD1	0.015634298324585	X			X		X		X	
HARD1	0.000729799270629883	X			X		X			X
HARD1	0.0306885242462158		X	X		X		X		
HARD1	0.0163125991821289		X	X		X			X	
HARD1	0.0155556201934814		X	X		X				X
HARD1	0.0156335830688477		X	X			X	X		
HARD1	0.0156328678131104		X	X			X		X	
HARD1	0		X	X			X			X
HARD1	0.0156307220458984		X		X	X		X		
HARD1	0.0156106948852539		X		X	X			X	
HARD1	0		X		X	X				X
HARD1	0		X		X		X	X		
HARD1	0.0312743186950684		X		X		X		X	
HARD1	0		X		X		X			X
Promedio:	0.0105305976337857		Mínimo:	0		Máximo:	0.0312743186950684			

Resultados para tablero HARD2

Sudoku	Tiempo	AC3	AC4	FU	MRV	UD	LCV	NI	FC	MAC
HARD2	0.0156326293945313			X		X		X		
HARD2	0.0156252384185791			X		X			X	
HARD2	0			X		X				X
HARD2	0			X			X	X		
HARD2	0.0163180828094482			X			X		X	
HARD2	0			X			X			X
HARD2	0				X	X		X		
HARD2	0.0150120258331299				X	X			X	
HARD2	0				X	X				X
HARD2	0.0156381130218506				X		X	X		
HARD2	0.000680446624755859				X		X		X	
HARD2	0				X		X			X
HARD2	0.0311918258666992	X		X		X		X		
HARD2	0.00452303886413574	X		X		X			X	
HARD2	0.0110049247741699	X		X		X				X
HARD2	0.0162734985351563	X		X			X	X		
HARD2	0.00111985206604004	X		X			X		X	
HARD2	0.0140056610107422	X		X			X			X
HARD2	0	X			X	X		X		
HARD2	0.015632152557373	X			X	X			X	
HARD2	0.0156247615814209	X			X	X				X
HARD2	0.00198507308959961	X			X		X	X		
HARD2	0.0140061378479004	X			X		X		X	
HARD2	0	X			X		X			X
HARD2	0.0312576293945313		X	X		X		X		
HARD2	0.0163474082946777		X	X		X			X	
HARD2	0.0167548656463623		X	X		X				X
HARD2	0		X	X			X	X		
HARD2	0.014012336730957		X	X			X		X	
HARD2	0.0156333446502686		X	X			X			X
HARD2	0		X		X	X		X		
HARD2	0.0156257152557373		X		X	X			X	
HARD2	0.0156242847442627		X		X	X				X
HARD2	0		X		X		X	X		
HARD2	0.0162856578826904		X		X		X		X	
HARD2	0		X		X		X			X
Promedio:	0.00960596402486165		Mínimo:	0		Máximo:	0.0312576293945313			

Para los tableros de 9x9, se obtienen resultados menores a 1 segundo para cualquier tablero, ya que se trabaja con una cantidad pequeña de datos y todas las técnicas pueden ser utilizadas e igualmente obtener un resultado en un corto período de tiempo.

6.1.1. Gráficos



En los gráficos de los experimentos realizados podemos observar que para el tablero EASY los datos están concentrados cerca de los 0.02 segundos, mientras que para el tablero HARD1 tienden a ser valores menores a 0.030 segundos. Para el tablero HARD2, ocurre lo mismo que el tablero HARD1, pero tienden a ser mayores los valores en comparación al tablero HARD1. Para el tablero EMPTY los datos tienden a estar muy cerca del valor 0, excepto por un valor atípico que sería el máximo para 0.109309673309326 segundos.

6.2. Tablero 16x16

Para el tablero vacío

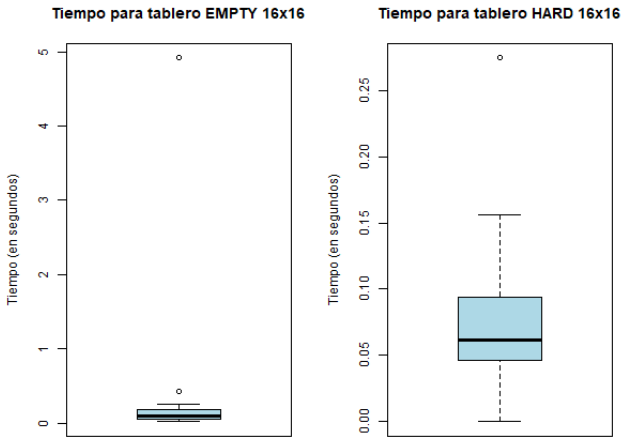
Sudoku	Tiempo	AC3	AC4	FU	MRV	UD	LCV	NI	FC	MAC
EMPTY	0.134868860244751			X		X		X		
EMPTY	0.0762321949005127			X		X			X	
EMPTY	0.0419073104858398			X		X				X
EMPTY	0.0312483310699463			X			X	X		
EMPTY	0.0628674030303955			X			X		X	
EMPTY	0.0312795639038086			X			X			X
EMPTY	0.0160140991210938				X	X		X		
EMPTY	0.0562970638275147				X	X			X	
EMPTY	0.0283522605895996				X	X				X
EMPTY	0.0310413837432861				X		X	X		
EMPTY	0.0777316093444824				X		X		X	
EMPTY	0.016118049621582				X		X			X
EMPTY	0.142212867736816	X		X		X		X		
EMPTY	0.0945558547973633	X		X		X			X	
EMPTY	0.138137102127075	X		X		X				X
EMPTY	0.0811905860900879	X		X			X	X		
EMPTY	0.148551464080811	X		X			X		X	
EMPTY	0.109374761581421	X		X			X			X
EMPTY	0.0609912872314453	X			X	X		X		
EMPTY	0.0875442028045654	X			X	X			X	
EMPTY	0.047358512878418	X			X	X				X
EMPTY	0.0321319103240967	X			X		X	X		
EMPTY	0.0696301460266113	X			X		X		X	
EMPTY	0.0471901893615723	X			X		X			X
EMPTY	4.91944980621338		X	X		X		X		
EMPTY	0.430402040481567		X	X		X			X	
EMPTY	0.237585067749023		X	X		X				X
EMPTY	0.194641351699829		X	X			X	X		
EMPTY	0.240064382553101		X	X			X		X	
EMPTY	0.193846225738525		X	X			X			X
EMPTY	0.144159078598022		X		X	X		X		
EMPTY	0.203281402587891		X		X	X			X	
EMPTY	0.152831792831421		X		X	X				X
EMPTY	0.14777684211731		X		X		X	X		
EMPTY	0.225672960281372		X		X		X		X	
EMPTY	0.174440860748291		X		X		X			X
Promedio:	0.247971634070079		Mínimo:	0.0160140991210938		Máximo:	4.91944980621338			

Para el tablero Hard16

Sudoku	Tiempo	AC3	AC4	FU	MRV	UD	LCV	NI	FC	MAC
HARD16	0.0706660747528076			X		X		X		
HARD16	0.136936664581299			X		X			X	
HARD16	0.0353646278381348			X		X				X
HARD16	0.0317277908325195			X			X	X		
HARD16	0.0628764629364014			X			X		X	
HARD16	0.0318517684936523			X			X			X
HARD16	0.0155060291290283				X	X		X		
HARD16	0.0468864440917969				X	X			X	
HARD16	0				X	X				X
HARD16	0.0156230926513672				X		X	X		
HARD16	0.051039457321167				X		X		X	
HARD16	0.0121827125549316				X		X			X
HARD16	0.156407594680786	X		X		X		X		
HARD16	0.0965306758880615	X		X		X			X	
HARD16	0.0617063045501709	X		X		X				X
HARD16	0.0475339889526367	X		X			X	X		
HARD16	0.0780239105224609	X		X			X		X	
HARD16	0.0786657333374023	X		X			X			X
HARD16	0.0448992252349854	X			X	X		X		
HARD16	0.0688791275024414	X			X	X			X	
HARD16	0.0471279621124268	X			X	X				X
HARD16	0.0485200881958008	X			X		X	X		
HARD16	0.0964970588684082	X			X		X		X	
HARD16	0.0372104644775391	X			X		X			X
HARD16	0.27451491355896		X	X		X		X		
HARD16	0.0942413806915283		X	X		X			X	
HARD16	0.0628879070281982		X	X		X				X
HARD16	0.125780582427979		X	X			X	X		
HARD16	0.0947287082672119		X	X			X		X	
HARD16	0.130385875701904		X	X			X			X
HARD16	0.0562450885772705		X		X	X		X		
HARD16	0.09438157081604		X		X	X			X	
HARD16	0.0463464260101318		X		X	X				X
HARD16	0.0468740463256836		X		X		X	X		
HARD16	0.0940957069396973		X		X		X		X	
HARD16	0.052811861038208		X		X		X			X
Promedio:	0.0707210368580288		Mínimo:	0		Máximo:	0.27451491355896			

Para el tablero Hard16 se obtiene el peor tiempo al aplicar las heurísticas de dominios no ordenados y de la primera variable no asignada, con el uso del algoritmo AC4. Para el tablero vacío, también ocurre lo mismo.
Mientras que los mejores tiempos para el tablero vacío toma en cuenta las heurísticas MRV, LCV y el algoritmo MAC. Para el tablero Hard16, también se tiene el mejor tiempo para las mismas heurísticas y el algoritmo MAC.
Podemos observar que tanto para el tablero vacío, como para el tablero Hard16 al utilizar el algoritmo AC4 se toma un tiempo mayor que para el algoritmo AC3.

6.2.1. Gráficos



En los gráficos de los experimentos realizados podemos observar que para el tablero EMPTY, la mayoría de los valores se concentra por debajo de 1 segundos y hay un valor atípico de 5 segundos. Para el tablero HARD los valores se concentran por debajo de los 0.15 segundos aproximadamente.

6.3. Tablero 25x25

Para el tablero vacío

Sudoku	Tiempo	AC3	AC4	FU	MRV	UD	LCV	NI	FC	MAC
EMPTY	1.32852935791016			X		X			X	
EMPTY	0.875848293304443			X		X				X
EMPTY	1.03397941589355			X			X	X		
EMPTY	0.297386407852173			X			X		X	
EMPTY	0.813549757003784			X			X			X
EMPTY	74.7616879940033				X	X			X	
EMPTY	0.40723991394043				X	X				X
EMPTY	0.0158367156982422				X		X	X		
EMPTY	0.235589504241943				X		X		X	
EMPTY	0.266029357910156				X		X			X
EMPTY	87.0157928466797	X		X		X		X		
EMPTY	1.22042918205261	X		X		X			X	
EMPTY	0.812689065933228	X		X		X				X
EMPTY	0.86089825630188	X		X			X	X		
EMPTY	1.1570360660553	X		X			X		X	
EMPTY	0.760063409805298	X		X			X			X
EMPTY	0.438113212585449	X			X	X		X		
EMPTY	0.841735363006592	X			X	X			X	
EMPTY	0.445020437240601	X			X	X				X
EMPTY	0.412262916564941	X			X		X	X		
EMPTY	0.800615310668945	X			X		X		X	
EMPTY	0.378515243530273	X			X		X			X
EMPTY	179.733966827393		X	X		X		X		
EMPTY	1.44021582603455		X	X		X			X	
EMPTY	0.930245161056519		X	X		X				X
EMPTY	0.963877201080322		X	X			X	X		
EMPTY	1.37469458580017		X	X			X		X	
EMPTY	0.924419641494751		X	X			X			X
EMPTY	0.564361810684204		X		X	X		X		
EMPTY	0.938955068588257		X		X	X			X	
EMPTY	0.582706928253174		X		X	X				X
EMPTY	0.549312591552734		X		X		X	X		
EMPTY	0.901769638061523		X		X		X		X	
EMPTY	0.542728424072266		X		X		X			X
Promedio:	10.7242971097722		Mínimo:	0.0158367156982422		Máximo:	179.733966827393			

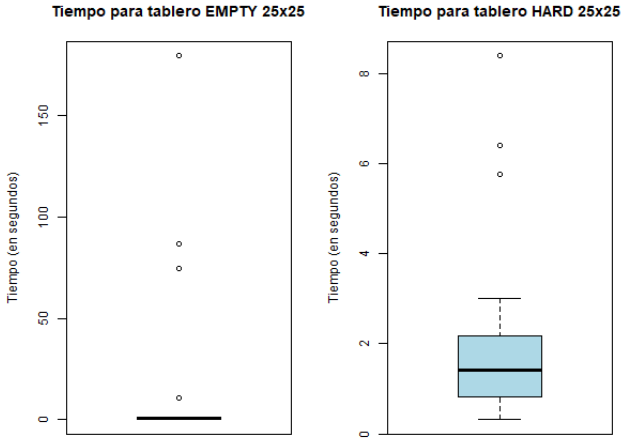
Podemos ver que para el tablero vacío de 25x25 el mínimo ocurre cuando se aplican las heurísticas MRV, LCV y se hace el backtracking.
Por otro lado, el máximo se alcanza cuando se hace uso de las heurísticas de primera variable no asignada, dominios no ordenados y se hace uso del algoritmo AC4.
Para este tablero podemos observar que los resultados que utilizan AC4 tienen un tiempo mayor a los que usan el algoritmo AC3, además al usar AC3 se requieren de heurísticas específicas, si se toma la primera variable no asignada o el dominio sin ningún orden, el algoritmo tiene un rendimiento deficiente.

Para el tablero difícil

Sudoku	Tiempo	AC3	AC4	FU	MRV	UD	LCV	NI	FC	MAC
HARD25	0.78253984451294			X		X			X	
HARD25	0.782927513122559			X		X				X
HARD25	1.20812368392944			X			X	X		
HARD25	0.990862846374512			X			X		X	
HARD25	0.818124771118164			X			X			X
HARD25	0.570156574249268				X	X			X	
HARD25	0.312459707260132				X		X	X		
HARD25	6.40840077400208				X		X		X	
HARD25	0.516061067581177				X		X			X
HARD25	2.30550169944763	X		X		X			X	
HARD25	2.18440103530884	X		X		X				X
HARD25	2.43576741218567	X		X			X	X		
HARD25	3.00766921043396	X		X			X		X	
HARD25	2.41447305679321	X		X			X			X
HARD25	1.37927055358887	X			X	X			X	
HARD25	0.830815076828003	X			X	X				X
HARD25	1.40973472595215	X			X		X	X		
HARD25	1.91033339500427	X			X		X		X	
HARD25	0.719223499298096	X			X		X			X
HARD25	8.4054217338562		X	X		X			X	
HARD25	5.76209592819214		X	X		X				X
HARD25	1.5973961353302		X	X			X	X		
HARD25	0.988449335098267		X	X			X		X	
HARD25	1.41031527519226		X	X			X			X
HARD25	1.82014060020447		X		X	X				X
HARD25	1.33570837974548		X		X		X	X		
HARD25	1.54510974884033		X		X		X		X	
HARD25	1.34134912490845		X		X		X			X
Promedio:	1.9711725967271		Mínimo:	0.312459707260132		Máximo:	8.4054217338562			

Por la tabla anterior podemos observar que se obtiene un tiempo menor al utilizar las heurísticas MRV, LCV y backtracking. Mientras que el que tiene el peor tiempo con el algoritmo AC4, las heurísticas de primera variable no asignada, dominios no ordenados y forward-checking.
Por los resultados obtenidos podemos apreciar que el forward-checking se beneficia de ciertas heurísticas como MRV y LCV, al ser utilizado junto a estas se obtienen resultados en poco tiempo igualmente al aplicar AC3 antes de correr el forward-checking, se mejoran los tiempos de ejecución.
De manera similar a lo obtenido para el tablero 16x16, el uso de AC4 tiene en promedio un tiempo de ejecución mayor al de AC3.

6.3.1. Gráficos



En los gráficos de los experimentos realizados podemos observar que para el tablero EMPTY, existen varios casos que superan los 10 segundos, como hemos visto en la tabla y para el tablero HARD la mayoría de los valores está concentrado entre 0 y 3 segundos, para los valores en la tabla. Por lo que los valores de tiempo de ejecución para HARD tienden a ser mayores.

7. Análisis de los resultados

Con los resultados obtenidos podemos observar que el uso del algoritmo backtracking con las heurísticas de MRV y LCV nos permiten obtener el resultado más rápido para Sudoku de un tamaño de hasta 25x25.
Los Sudokus vacíos son las versiones de menor dificultad para cualquiera de los algoritmos que se han utilizado. Pero a pesar de esto, al correr el algoritmo de backtracking con la heurística de primera variable no asignada, dominios no ordenados y AC4 se obtiene uno de los peores resultados para el caso 16x16. Algo similar ocurre para el caso 9x9, incluso al utilizar las heurísticas de primera variable no asignada, valores del dominio no ordenados y sin forward-checking, se obtiene el peor tiempo de ejecución con el uso del algoritmo AC4.

Por los resultados obtenidos, podemos ver que AC3 es una mejor opción para la solución del problema de Sudoku. También es posible afirmar por los resultados obtenidos que el forward-checking es de gran utilidad para los casos de Sudoku que tienen mayor tamaño, pero en el caso de la primera variable no asignada, esta heurística no es útil, debido a que no nos permite disminuir la menor cantidad posible de recursiones del backtracking, lo que tiene un gran impacto para los Sudokus de mayor tamaño.
Por otro lado, el algoritmo MAC tuvo un buen rendimiento para Sudokus de gran tamaño, no fue el algoritmo que permitió obtener mejores resultados pero son resultados lo suficientemente rápidos. A pesar de que, según Wallace, para los experimentos realizados la heurística Domjup utilizada otorga un mejor rendimiento debido a menores cantidades de restricciones, para los Sudokus no da los mejores resultados en las pruebas realizadas pero el tiempo de ejecución es corto para este algoritmo.

8. Conclusiones y recomendaciones

Una vez realizados los experimentos para Sudokus de 9x9, 16x16 y 25x25 podemos afirmar que la mejor solución para Sudokus de cualquier tamaño viene dada por los algoritmos backtracking con el algoritmo MAC y las heurísticas MRV y LCV. El algoritmo MAC es una mejora del AC3, pero también se pueden obtener resultados rápidos por medio de AC3 y el uso de las heurísticas anteriomente mencionadas.
Para Sudokus con tamaño 9x9 no existe una gran diferencia entre los tiempos de ejecución, ya que estamos trabajando con un volúmen de datos pequeño, mientras que para el caso de 16x16 y 25x25 es muy importante el uso de los algoritmos correctos para obtener resultados en un período de tiempo razonable.
Al igual que los resultados obtenidos por Wallace, en el caso del Sudoku, el algoritmo AC3 es capaz de contribuir a obtener resultados en una cantidad menor de tiempo, gracias a que elimina los valores del dominio de cada variable que no sea consistente con las restricciones que han sido establecidas. Y el algoritmo AC3 es una mejor opción para la resolución de Sudokus, en comparación al algoritmo AC4, que no permite obtener una solución tan rápida y empeora con el aumento del número de filas y columnas en el tablero.

9. Referencias

Wallace, R (1993). Why AC-3 is Almost Always Better than AC-4 for Establishing Arc Consistency in CSPs. Department of Computer Science, University of New Hampshire Durham, NH 03824, U. S. A.

https://www.ijcai.org/Proceedings/93-1/Papers/034.pdf

Peter Norvig et al. AIMA library. https://github.com/aimacode/aima-python

Capítulo 10. Problema de satisfacción de restricciones (CSP). https://personales.upv.es/misagre/papers/capitulo.pdf

Russell, S. J. 1., & Norvig, P. (2003). Artificial intelligence: a modern approach. Prentice Hall.