Third Year Project 2013/2014

Report 5 Constraint Satisfaction, particularly consistency techniques.

A Sudoku Solver Using Constraint Satisfaction (& Other Techniques)

Thomas Paul Clarke

A report submitted in part fulfilment of the degree of

BSc (Hons) in Computer Science

Supervisor: Dave Cohen

A constraint satisfaction problem is as specific problem that is best solved using strategies that relate to constraints. These are the limitations that are put on an object, the process of creating an object is also linked to what that object will be able to do to validate its value or a value it could take on. There are many ways to provide constraints, it is essentially a set of rules that the objects associated with it must conform to, this will provide the actions needed with the boolean outcomes from these operations.

The use of a constraint satisfaction problems is often linked to logic puzzles like Eight Queens and Sudoku, this is because the games rules are all about limiting what is placed and its location due it collisions or duplicate numbers in these cases. There are many different ways to apply constraint satisfaction and each program needs to have all of the rules in place so that it can be applied correctly.

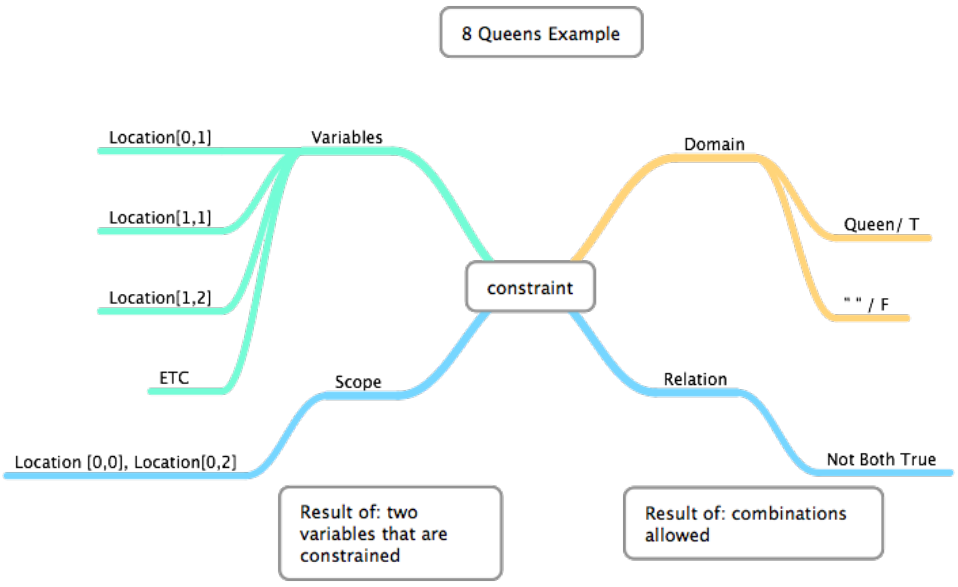
A constraint[1] is split into two parts, the variables and the domain, the variables are the questions that is to be answered, essentially in these programs it will be a location on a board that will need to be checked if it is a correct state or a false. The domain is the other part to the constraint and contains a set of values that are the allowed for the variables. If the variables within the constraint are also within the domain then the constraint is satisfied.

[1] Professor David Cohen. The Complexity of the Constraint Satisfaction Problem. ISG-CS Mini-Conference Slides
http://www.rhul.ac.uk/computerscience/documents/pdf/csisgmini-conf/dac.pdf

Each constraint has variables and in this example the variable is a Queen from the 8 queen puzzle, in this case there is a variable that is the location the queen has been placed and now needs to be checked, this is done using constraint satisfaction by using the domain this is given with the variable and contains the value that the Queen can take, if satisfied it can be placed as a Q otherwise the cell will be left empty.

Within the constraint there is a scope, this is the relation between two variables that are constrained that have come from the tuple, which assigns the variables to the scope. This is when the two constraints are matched together and called a scope and are then constrained to produce the relation. The relation uses the scope currently used and is the relation between them in terms of their values and the allowed combination between them.

The diagram below shows this process related to the 8 Queen puzzle as an example to show the way P = (Variable, Domain, Constraint)



Within constraint satisfaction there are techniques that can be used to increase efficiency, however these are deterministic which means that unlike a search it will provide the same results each time, given a particular input there will always be a constant output.[2] This proves to be very useful as it allows the computation to be made as soon as possible. [3]

[2] http://technet.microsoft.com/en-us/library/aa214775(v=sql.80).aspx
[3] Discrete Mathematics & its Applications, 6th Edition. by Kenneth H. Rosen. Page 836

To improve the constraint it is possible to select the objects that are no longer needed due to a variable being domain consistent, meaning that within the domain of the variable contains a value that is matched within the constraints. [4]This means that this branch and any that it continues to follow are not needed, as they are not valid. To improve this situation pruning the domains as much as possible is needed as it allows the efficiency advantage without creating impossible boards.

A method I have used a method called backtracking which unlike forward checking and arc consistency which will be discussed later it does not have any propagation, meaning that there is no reproductions of the states, it is the same board that is edited and deleted to find the desired state. Backtracking works by searching and if the outcome matches the desired output then it passes.

Backtracking is a recursive algorithm used to find a desired output through using the brute force approach, it does this by starting at the root of the tree and you follow a consistent path until the end of that path within a leaf, this is then when the leaf is evaluated into the correct output where the algorithm stops and the value has been found, or an incorrect node is found and then this process is repeated by going back to the parent of the leaf and following another node and check that one, this is done until the full tree has been traversed and there are no correct nodes on that tree, or that a correct node is found and the recursion stops. [5]

However Arc Consistency is different the domains are reduces and there are propagates of each state. The Generalised Arc Consistency algorithm is used to simplify the constraint satisfaction problem by continually iterating through the constraints and their implications of the variables one at a time. The difference between the generalised arc consistently and arc consistency is the use of non binary constraints using tuples of variables instead of a pair. Here is the algorithm and this will be commented and described line by line at what it does.

[4] http://www.cs.ubc.ca/~kevinlb/teaching/cs322%20-%202009-10/Lectures/CSP3.pdf
[5] http://www.cis.upenn.edu/~matuszek/cit594-2012/Pages/backtracking.html

Generalized arc consistency algorithm[6]

1: **Procedure** GAC (Variables, Domain, Constraints) [7]
2: **Inputs**
3: *V* // a set of variables
4: *domain*// a function such that *domain (X)* is the domain of variable *X*
5: *C* // set of constraints to be satisfied
6: **Output**
7: //arc-consistent domains for each variable to cut down domains that increase efficiency.
8: **Local**
9: $\boldsymbol{D}_X$ //is a set of values for each variable *X*
10: *TDA* // is a set of arcs
$Domain_{ABC}$ = {1,2,3,4}
TDA = {(A, A<B), (B, A<B), (C, B<C)}
11: **for each** variable *X* **do**
12: $\boldsymbol{D}_X \leftarrow domain(X)$
13: $TDA \leftarrow \{\langle X,c\rangle | \ c \in C \ \text{and} \ X \in scope(c)\}$ // For all the variables create the scopes, each variable within the domain creates the scope.
$Scope_A$ = {A,B} $Scope_B$ = {A,B} $Scope_C$ = {B,C}
14: **while** ($TDA \neq \{\}$) // While the TDA set is not empty, meaning all the constraints have not been satisfied. When they have stop.
15: **select** $\langle X,c\rangle \in TDA$; // For each constraint within the TDA
16: $TDA \leftarrow TDA \setminus \{\langle X,c\rangle\}$;

Step 1:
Select = (A, A<B)
$Domain_A$ = {1,2,3,4}
After Step 2:
$Domain_A$ = {1,2,3}
TDA = {~~(A, A<B),~~ (B, A<B), (C, B<C)}

Step 3:
Select = (B, A<B)
$Domain_B$ = {1,2,3,4}
After Step 4:
$Domain_{B2}$ = {2,3,4}
TDA = {~~(A, A<B),~~ ~~(B, A<B),~~ (C, B<C)}
Goto Step 5: (line 18-20)

Step 7:
Select = (A, A<B)
$Domain_A$ = {1,2,3,}
After Step 8:
$Domain_{A2}$ = {1,2}
TDA = {~~(A, A<B), (B, A<B),~~ (C, B<C)}
Goto Step 9: (line 18-20)

Step 11:
Select = (C, B<C)
$Domain_C$ = {1,2,3,4}

[6] http://artint.info/html/ArtInt_79.html
[7] http://www.youtube.com/watch?v=Ge-XiX_tP4c - Doug Fisher's video on Generalized Arc Consistency

After Step 12:
$Domain_{C2}$ = {2,3,4}
TDA = {~~{A, A<B)~~, ~~{B, A<B)~~, ~~{C, B<C)}~~
Goto Step 13: (line 18-20)

Step 15
TDA set empty, all must be reduced

17: $ND_X \leftarrow \{x/\ x \in \boldsymbol{D}_X$ and some $\{X=x, Y_1=y_1,...,Y_k=y_k\} \in c$ where $y_i \in \boldsymbol{D}_{Yi}$ for all $i\}$

Step 2:
A = 4 (Did not pass test)
$Domain_{A2}$ = {1,2,3}

Step 4:
B = 1 (Did not pass test)
$Domain_{B2}$ = {2, 3, 4}

Step 8:
A = 3 (Did not pass test)
$Domain_{B2}$ = {1, 2}

Step 12:
C = 1 (Did not pass test)
$Domain_{C2}$ = {2, 3, 4}

18: **if** ($ND_X \neq \boldsymbol{D}_X$) **then** // if the new domain is not the same as the old domain due to it being reduced.

19: $TDA \leftarrow TDA \cup \{\langle Z,c'\rangle | X \in scope(c'), c'$ is not $c, Z \in scope(c') \setminus \{X\}\}$ //
Within the TDA set now that the domain has changed it needs to be considered again so that it can be checked due to the new domain changes within a constraint that is in its scope.

Step 5
(A, A<B) with $Domain_{A2}$ = {1, 2, 3}
Compare
(B, A<B) with $Domain_{B2}$ = {2, 3, 4}
False - (A, A<B) $Domain_{A2}$ needs to be {1,2} Goto Step 6

Step 9
(A, A<B) with $Domain_{A2}$ = {1, 2}
Compare
(B, A<B) with $Domain_{B2}$ = {2, 3, 4}
True - (A, A<B) $Domain_{A2}$ is {1,2} Goto Step 10

Step 13
(B, A<C) with $Domain_{B2}$ = {2, 3, 4}
Compare
(C, B<C) with $Domain_{B2}$ = {2,3,5}
false - (A, A<B) $Domain_{A2}$ is {2,3,5} Goto Step 10 // This is repeated until
true - (A, A<B) $Domain_{A2}$ is {5} Goto Step 14

20: $\boldsymbol{D}_X \leftarrow ND_X$

Step 6
(A, A<B) $Domain_{A1}$ = {1,2,3}
return to top of loop

Step 10
(A, A<B) $Domain_{A1}$ = {1,2}
return to top of loop

Step 10
(A, A<B) $Domain_{B1}$ = {2,3,4}
return to top of loop

Step 14
(A, A<B) $Domain_{C1}$ = {4}
return to top of loo

21: **return** $\{\boldsymbol{D}_X / X$ *is a variable*$\}$

The reason to go through this algorithm was to show what each step does and how it relates to some real values, this example shows how 3 different constraints are

used on a domain and how they effect each one with the variables within it. Showing the steps of the algorithm and how it is intended to reduce each domain that are not then used within the next part of the program. This can relate to the Sudoku solver, as there are several different ways to create each board and store them, and then trim the branches of the boards that are not valid due to duplications. However this process can cause a lot of processing, as each board will have to be tested until there is one that returns false and that branch is trimmed, but have still been produced.

This algorithm prevents this by not allowing redundant boards to be created as the given domains are reduced to prevent propagation, meaning there will not be duplicates when a tree of boards are created. Instead of trimming the tree this process has prevented those from being created therefore increasing the efficiency of the program.

Another constraint propagation algorithm the Dynamic Variable Ordering process uses the most constrained variable to be the first one used, this has a purpose as it allows the dead ends to be discovered as early as possible and without the need to be already considered from other variables, this effective heuristic cuts down the search space. [8]

Dynamic ordering means the choice of the next variable that is used will be dependant on the state of the search, this is found using forward checking, which is the opposite to backtracking, This means that the current domain is dependant on the previous and current constraints. The reason for this is to use the best possible variable in order to increase the efficiency.[9]

The concept of dynamic variable ordering is to compare all the available variables and control which constraints will be compared first, this is done by using the variable that has the most constraints, if the number of constraints are high then it is likely to produce the highest number of fails when tested as it allows the most constrained. The point of this is to get as many of the failed tests done in the early stages so that they do not have to be replicated later. This will improve the efficiency when searching and creating the rest of the board. [10]

[8] www.ics.uci.edu/~dechter/courses/ics-275a/spring.../chapter5-09.ppt slides 26-31
[9] http://ktiml.mff.cuni.cz/~bartak/constraints/ordering.html
[10] http://www.dcs.gla.ac.uk/~pat/cp4/papers/95_14.pdf

This could benefit the Sudoku solver as it will allow the board states to be compared and searched in an effective way so that the states that will fail are found as early as possible. It will be used when the next cell needs to be populated within the grid, using the tightest constraints on the cell to produce the failed states, for example:

Cell = variable

Column = {2,4,3}

Row = {1,2,3,4,5,6,7,8}

Square = {1,2}

```
For loop {
Constraint 1 = ( cell.value != row.cell[i] && cell.value != column.cell[i])
Constraint 2 = (cell.value != square.cell[i])
}
```

Constraint1 have more constraints within it so the dynamic variable ordering will use the variable that has the most constraints. This will allow the most tests to be done first using the constraints, failing the tests means that that each time the possibility of that option is ruled out, this could mean that the more restrictive test (constraint1) can be done and constraint 2 might not even need to be considered. [11]

Constraint satisfaction is essential within the Sudoku solver, it will allow the process of handling the boards in terms of finding the next cell by using a tree that contains the options, this however will have to be restricted by using these algorithms to make the searching more effective and reduce the options to only valid boards and not include boards that are the same.

[11] http://www.dcs.gla.ac.uk/~pat/cp4/papers/95_14.pdf