Assignment Lecture 3: Constraint Satisfaction Problems

Håkon Måløy

October 1, 2021

Constraint Satisfaction Problems (CSPs)

Constraint Satisfaction Problems (CSPs)

We have seen how searching can solve problems:

Constraint Satisfaction Problems (CSPs)

We have seen how searching can solve problems:

- Use domain specific heuristic.

Constraint Satisfaction Problems (CSPs)

We have seen how searching can solve problems:

- Use domain specific heuristic.
- Each state is atomic (atomic representation).

Constraint Satisfaction Problems (CSPs)

We have seen how searching can solve problems:

- Use domain specific heuristic.
- Each state is atomic (atomic representation).
  - Domain specific code is needed to describe transitions between states.

Constraint Satisfaction Problems (CSPs)

We have seen how searching can solve problems:

- Use domain specific heuristic.
- Each state is atomic (atomic representation).
  - Domain specific code is needed to describe transitions between states.

CSPs describe states using a factored representation:

Constraint Satisfaction Problems (CSPs)

We have seen how searching can solve problems:

- Use domain specific heuristic.
- Each state is atomic (atomic representation).
  - Domain specific code is needed to describe transitions between states.

CSPs describe states using a factored representation:

- A set of variables.

Constraint Satisfaction Problems (CSPs)

We have seen how searching can solve problems:

- Use domain specific heuristic.
- Each state is atomic (atomic representation).
  - Domain specific code is needed to describe transitions between states.

CSPs describe states using a factored representation:

- A set of variables.
- Each variable has a value.

Constraint Satisfaction Problems (CSPs)

We have seen how searching can solve problems:

- Use domain specific heuristic.
- Each state is atomic (atomic representation).
  - Domain specific code is needed to describe transitions between states.

CSPs describe states using a factored representation:

- A set of variables.
- Each variable has a value.

A problem is solved when each variable has a value that satisfies the constraints on that variable.

Constraint Satisfaction Problems (CSPs)

Constraint Satisfaction Problems (CSPs)

Formally defining CSPs:

Constraint Satisfaction Problems (CSPs)

Formally defining CSPs:

- A set of variables $\mathcal{X}$:
  - They can be thought of as memory slots that can be assigned a value. e.g. $A : \square, B : \square, C : \square$.

Constraint Satisfaction Problems (CSPs)

Formally defining CSPs:

- A set of variables $\mathcal{X}$:
- A set of domains $\mathcal{D}$:

- They can be thought of as memory slots that can be assigned a value. e.g. $A : \square, B : \square, C : \square$.
- The set of values that can be assigned to a variable. e.g. $\{1, 2, 3, 4\}$.

Constraint Satisfaction Problems (CSPs)

Formally defining CSPs:

- A set of variables $\mathcal{X}$:
- A set of domains $\mathcal{D}$:
- A set of constraints $\mathcal{C}$:

- They can be thought of as memory slots that can be assigned a value. e.g. $A : \square, B : \square, C : \square$.
- The set of values that can be assigned to a variable. e.g. $\{1, 2, 3, 4\}$.
- A set of restrictions on one, or more, of the variables. e.g. $\{\mathrm{A} \neq \mathrm{B}\}$.

Constraint Satisfaction Problems (CSPs)

Formally defining CSPs:

- A set of variables $\mathcal{X}$:
- A set of domains $\mathcal{D}$:
- A set of constraints $\mathcal{C}$:
- A solution is both consistent and complete.

- They can be thought of as memory slots that can be assigned a value. e.g. $A : \square, B : \square, C : \square$.
- The set of values that can be assigned to a variable. e.g. $\{1, 2, 3, 4\}$.
- A set of restrictions on one, or more, of the variables. e.g. $\{A \neq B\}$.
- All constraints are met and all variables have been assigned a value.

Different Constraint Types

Different Constraint Types

- Unary:
- Restricts the value of a single variable. e.g. $A \neq 1$

Different Constraint Types

- Unary:
- Binary:

- Restricts the value of a single variable. e.g. $A \neq 1$
- Restricts the values of two variables. e.g. $A \neq B$

Different Constraint Types

- Unary:
- Binary:
- Global:

- Restricts the value of a single variable. e.g. $A \neq 1$
- Restricts the values of two variables. e.g. $A \neq B$
- Restricts an arbitrary number of variables. e.g. $Alldiff(A, B, C, D, ..., Z)$

Different Constraint Types

- Unary:
- Binary:
- Global:

- Restricts the value of a single variable. e.g. $A \neq 1$
- Restricts the values of two variables. e.g. $A \neq B$
- Restricts an arbitrary number of variables. e.g. $Alldiff(A, B, C, D, ..., Z)$

We can encode global constraints as binary constraints:

Different Constraint Types

- Unary:
- Binary:
- Global:

- Restricts the value of a single variable. e.g. $A \neq 1$
- Restricts the values of two variables. e.g. $A \neq B$
- Restricts an arbitrary number of variables. e.g. $Alldiff(A, B, C, D, ..., Z)$

We can encode global constraints as binary constraints:

- $Alldiff(A, B, C, D, ..., Z) =$ $Diff(A, B), Diff(A, C) ..., Diff(Z, Y)$.

Different Constraint Types

- Unary:
- Binary:
- Global:

- Restricts the value of a single variable. e.g. $A \neq 1$
- Restricts the values of two variables. e.g. $A \neq B$
- Restricts an arbitrary number of variables. e.g. $Alldiff(A, B, C, D, ..., Z)$

We can encode global constraints as binary constraints:

- $Alldiff(A, B, C, D, ..., Z) = Diff(A, B), Diff(A, C) ..., Diff(Z, Y)$.
- Why?

Different Constraint Types

- Unary:
- Binary:
- Global:

- Restricts the value of a single variable. e.g. $A \neq 1$
- Restricts the values of two variables. e.g. $A \neq B$
- Restricts an arbitrary number of variables. e.g. $Alldiff(A, B, C, D, ..., Z)$

We can encode global constraints as binary constraints:

- $Alldiff(A, B, C, D, ..., Z) =$ $Diff(A, B), Diff(A, C) ..., Diff(Z, Y)$.
- Why?
  - Some efficient algorithms expect binary constraints.

Encoding a Meal with Friends as a CSP

Encoding a Meal with Friends as a CSP

Four friends, 3 types of cheese, 2 types of wine, Friend C does not like blue cheese, Friend B does not like red wine with cheese.

Encoding a Meal with Friends as a CSP

Four friends, 3 types of cheese, 2 types of wine, Friend C does not like blue cheese, Friend B does not like red wine with cheese.

- $\mathcal{X} : \{C1, C2, C3, W1, W2\}$

Encoding a Meal with Friends as a CSP

Four friends, 3 types of cheese, 2 types of wine, Friend C does not like blue cheese, Friend B does not like red wine with cheese.

- $\mathcal{X} : \{C1, C2, C3, W1, W2\}$
- $\mathcal{D} : \{\mathcal{D}_{C1} : \{\forall cheese\}, \mathcal{D}_{C2} : \{\forall cheese\}, \mathcal{D}_{C3} : \{\forall cheese\}, \mathcal{D}_{W1} : \{\forall wine\}, \mathcal{D}_{W2} : \{\forall wine\}\}$

Encoding a Meal with Friends as a CSP

Four friends, 3 types of cheese, 2 types of wine, Friend C does not like blue cheese, Friend B does not like red wine with cheese.

- $\mathcal{X} : \{C1, C2, C3, W1, W2\}$
- $\mathcal{D} : \{\mathcal{D}_{C1} : \{\forall cheese\}, \mathcal{D}_{C2} : \{\forall cheese\}, \mathcal{D}_{C3} : \{\forall cheese\}, \mathcal{D}_{W1} : \{\forall wine\}, \mathcal{D}_{W2} : \{\forall wine\}\}$
- $\mathcal{C} : \{\langle C1, C1 \neq blue\rangle, \langle C2, C2 \neq blue\rangle, \langle C3, C3 \neq blue\rangle, \langle (C1, C2, C3), Alldiff(C1, C2, C3)\rangle \langle W1, W1 \neq red\rangle, \langle W1, W1 \neq red\rangle, \langle (W1, W2), Diff(W1, W2)\rangle\}$

Encoding Soduku as a CSP

	A	B	C	D	E	F	G	H	I
1	5	3			7				
2	6			1	9	5			
3		9	8					6	
4	8				6				3
5	4			8		3			1
6	7				2				6
7		6					2	8	
8				4	1	9			5
9					8			7	9

- $\mathcal{X}$:
- $\mathcal{D}$:
- $\mathcal{C}$:

Arc Consistency

Arc Consistency

A variable X_i is arc consistent with another variable X_j if:

Arc Consistency

A variable X_i is arc consistent with another variable X_j if:

- For every value in the current domain $\mathcal{D}_i$

Arc Consistency

A variable X_i is arc consistent with another variable X_j if:

- For every value in the current domain $\mathcal{D}_i$
- There is some value in domain $\mathcal{D}_j$

Arc Consistency

A variable X_i is arc consistent with another variable X_j if:

- For every value in the current domain $\mathcal{D}_i$
- There is some value in domain $\mathcal{D}_j$
- That satisfies the binary constraint on the arc (X_i, X_j)

Arc Consistency

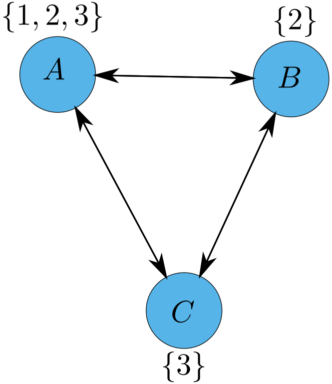
A variable X_i is arc consistent with another variable X_j if:

- For every value in the current domain $\mathcal{D}_i$
- There is some value in domain $\mathcal{D}_j$
- That satisfies the binary constraint on the arc (X_i, X_j)

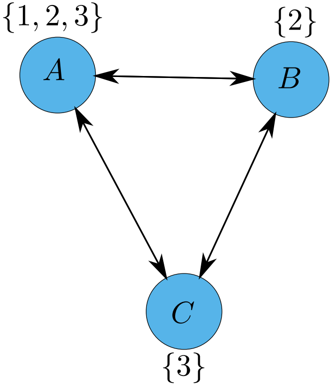
The AC-3 algorithm makes every variable arc consistent

AC-3 - Example

AC-3 - Example

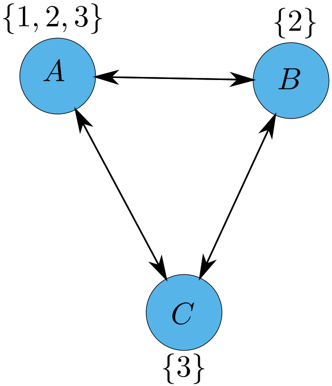


AC-3 - Example



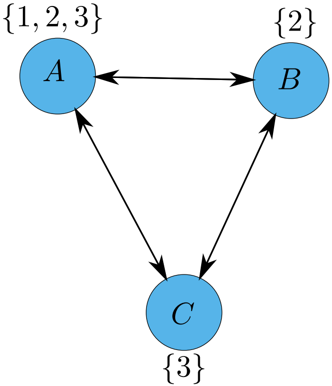
- $\mathcal{X} : \{A, B, C\}$

AC-3 - Example



- $\mathcal{X} : \{A, B, C\}$
- $\mathcal{D} : \{A : \{1, 2, 3\}, B : \{2\}, C : \{3\}\}$

AC-3 - Example



- $\mathcal{X} : \{A, B, C\}$
- $\mathcal{D} : \{A : \{1, 2, 3\}, B : \{2\}, C : \{3\}\}$
- $\mathcal{C} : \mathit{Alldiff}(A, B, C)$

AC-3 - Pseudocode

AC-3 - Pseudocode

```
function AC-3(csp)
    queue ← a queue of arcs/edges, initially all the arcs in the csp
    while queue is not empty do
        (X_i, X_j) ← POP(queue)
        if REVISE(csp, X_i, X_j) then
            if size of D_i = 0 then return false
            for each X_k in X_i.NEIGHBORS − {X_j} do
                add (X_k, X_j) to queue
    return true
function REVISE(csp, X_i, X_j)
    revised ← false
    for each x in D_i do
        if no value y in D_j allows (x, y) to satisfy the constraint between X_i and X_j then
            delete x from D_i
            revised ← true
    return revised
```

AC-3 Problem

AC-3 Problem

AC-3 only detects inconsistencies between binary constraints:

AC-3 Problem

AC-3 only detects inconsistencies between binary constraints:

- It detects constraints between $\{A \leftrightarrow B\}$

AC-3 Problem

AC-3 only detects inconsistencies between binary constraints:

- It detects constraints between $\{A \leftrightarrow B\}$
- It does not detect constraints $\{A \leftrightarrow B \leftrightarrow C\}$

AC-3 Problem

AC-3 only detects inconsistencies between binary constraints:

- It detects constraints between $\{A \leftrightarrow B\}$
- It does not detect constraints $\{A \leftrightarrow B \leftrightarrow C\}$

We've now made the variables arc consistent, but what if there are still multiple values possible for variables?

Backtracking Search

Backtracking Search

- Choose an unassigned variable.

Backtracking Search

- Choose an unassigned variable.
- Try all values in the domain in turn.

Backtracking Search

- Choose an unassigned variable.
- Try all values in the domain in turn.
- Recursively call itself to (hopefully) find a solution.

Backtracking Search

- Choose an unassigned variable.
- Try all values in the domain in turn.
- Recursively call itself to (hopefully) find a solution.
  - If the call succeeds: return solution.

Backtracking Search

- Choose an unassigned variable.
- Try all values in the domain in turn.
- Recursively call itself to (hopefully) find a solution.
  - If the call succeeds: return solution.
  - If it fails: restore assignment to previous state and try next value.

Backtracking Search

Backtracking Search

- A Depth-Limited Search.

Backtracking Search

- A Depth-Limited Search.
- Since CSPs are commutative the order of assignment of values to variables is not important.

Backtracking Search

- A Depth-Limited Search.
- Since CSPs are commutative the order of assignment of values to variables is not important.
- We consider only one variable per level in the tree.

Backtracking Search

- A Depth-Limited Search.
- Since CSPs are commutative the order of assignment of values to variables is not important.
- We consider only one variable per level in the tree.
- At each level in the tree we must decide which variable to deal with.

Backtracking Search - Pseudocode

Backtracking Search - Pseudocode

```
function BACKTRACKING-SEARCH(csp) return BACKTRACK(csp, {})
function BACKTRACK(csp, assignment)
    if assignment is complete then return assignment
    var ← SELECT-UNASIGNED-VARIABLE(csp, assignment)
    for each value in ORDER-DOMAIN-VALUES(csp, var, assignment) do
        if value is consistent with assignment then
            add {var = value} to assignment
            inferences ← INFERENCE(csp, var, assignment)
            if inferences ≠ failure then
                add inferences to csp
                result ← BACKTRACK(csp, assignment)
                if result ≠ failure then return result
                remove inferences from csp
          remove var = value from assignment
    return failure
```

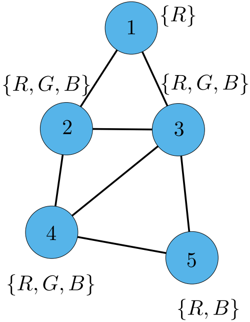
Backtracking Search - Exam Example

Backtracking Search - Exam Example

Consider the following constraint graph (in the figure) for a graph coloring problem where the constraints mean that the connected nodes cannot have the same color. The variables are shown inside the nodes while the domains are shown next to each variable node.

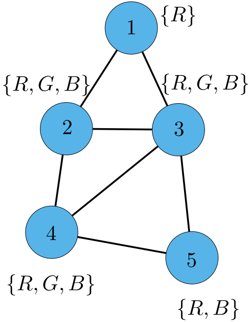
Backtracking Search - Exam Example

Consider the following constraint graph (in the figure) for a graph coloring problem where the constraints mean that the connected nodes cannot have the same color. The variables are shown inside the nodes while the domains are shown next to each variable node.



Backtracking Search - Exam Example

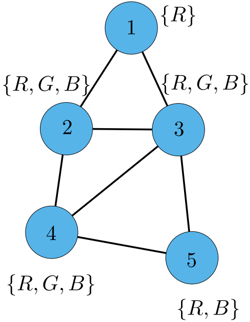
Consider the following constraint graph (in the figure) for a graph coloring problem where the constraints mean that the connected nodes cannot have the same color. The variables are shown inside the nodes while the domains are shown next to each variable node.



- What are the domains after a full constraint propagation using an arc consistency algorithm?

Backtracking Search - Exam Example

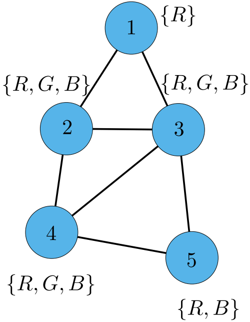
Consider the following constraint graph (in the figure) for a graph coloring problem where the constraints mean that the connected nodes cannot have the same color. The variables are shown inside the nodes while the domains are shown next to each variable node.



- What are the domains after a full constraint propagation using an arc consistency algorithm?
- Show the sequence of variable assignments during a pure backtracking search (don't assume that propagation above has been done). Assume that the variables are examined in numerical order and the values are assigned in the order shown next to each node. Show assignments by writing the variable number and the letter for the value, e.g, 5R, 2G.

Backtracking Search - Exam Example

Consider the following constraint graph (in the figure) for a graph coloring problem where the constraints mean that the connected nodes cannot have the same color. The variables are shown inside the nodes while the domains are shown next to each variable node.



- What are the domains after a full constraint propagation using an arc consistency algorithm?
- Show the sequence of variable assignments during a pure backtracking search (don't assume that propagation above has been done). Assume that the variables are examined in numerical order and the values are assigned in the order shown next to each node. Show assignments by writing the variable number and the letter for the value, e.g, 5R, 2G.
- This time you'll apply backtracking search with forward checking. Use the same ordering convention for variables and values as above. Show the sequence of variable assignments during backward search with forward checking. Again, show assignments by writing the number of variables followed by the letter for the value.

Assignment 4 - Description

Assignment 4 - Description

You will be playing sudoku!

	A	B	C	D	E	F	G	H	I
1	5	3			7				
2	6			1	9	5			
3		9	8					6	
4	8				6				3
5	4			8		3			1
6	7				2				6
7		6					2	8	
8				4	1	9			5
9					8			7	9

Assignment 4 - Description

You are provided with skeleton code that represents $\mathcal{X}, \mathcal{D}$ and $\mathcal{C}$ as follows:

Assignment 4 - Description

You are provided with skeleton code that represents $\mathcal{X}, \mathcal{D}$ and $\mathcal{C}$ as follows:

- $\mathcal{X}$: An array of variable names. e.g. $[var1, var2, var3, ...]$

Assignment 4 - Description

You are provided with skeleton code that represents $\mathcal{X}, \mathcal{D}$ and $\mathcal{C}$ as follows:

- $\mathcal{X}$: An array of variable names. e.g. $[var1, var2, var3, ...]$
- $\mathcal{D}$: A hash table/dictionary that associates a variable name with its legal values. e.g. $\{var1 : [1, 3, 8, 0]\}$

Assignment 4 - Description

You are provided with skeleton code that represents $\mathcal{X}, \mathcal{D}$ and $\mathcal{C}$ as follows:

- $\mathcal{X}$: An array of variable names. e.g. $[var1, var2, var3, ...]$
- $\mathcal{D}$: A hash table/dictionary that associates a variable name with its legal values. e.g. $\{var1 : [1, 3, 8, 0]\}$
- $\mathcal{C}$: A hash table that associates a variable name i with another hash table, where this second hash table contains the constraints affecting variable i. Specifically, the second hash table associates the name of another variable $j (i \neq j)$ with an array of legal pairs of values for the pair of variables (i, j).

Assignment 4 - Deliverables

Assignment 4 - Deliverables

The deliverables for the assignment are:

Assignment 4 - Deliverables

The deliverables for the assignment are:

- Well commented source code for a CSP solver based on backtracking search and AC-3, that is able to solve Sudoku boards.

Assignment 4 - Deliverables

The deliverables for the assignment are:

- Well commented source code for a CSP solver based on backtracking search and AC-3, that is able to solve Sudoku boards.
- Your program's solution for each of the four boards in the assignment.

Assignment 4 - Deliverables

The deliverables for the assignment are:

- Well commented source code for a CSP solver based on backtracking search and AC-3, that is able to solve Sudoku boards.
- Your program's solution for each of the four boards in the assignment.
- The number of times your BACKTRACK function was called, and the number of times your BACKTRACK function returned *failure* for each of the four boards. Comment briefly on these numbers for each of the four boards. E.g.:

Assignment 4 - Deliverables

The deliverables for the assignment are:

- Well commented source code for a CSP solver based on backtracking search and AC-3, that is able to solve Sudoku boards.
- Your program's solution for each of the four boards in the assignment.
- The number of times your BACKTRACK function was called, and the number of times your BACKTRACK function returned *failure* for each of the four boards. Comment briefly on these numbers for each of the four boards. E.g.:
  - What do the numbers mean?

Assignment 4 - Deliverables

The deliverables for the assignment are:

- Well commented source code for a CSP solver based on backtracking search and AC-3, that is able to solve Sudoku boards.
- Your program's solution for each of the four boards in the assignment.
- The number of times your BACKTRACK function was called, and the number of times your BACKTRACK function returned *failure* for each of the four boards. Comment briefly on these numbers for each of the four boards. E.g.:
  - What do the numbers mean?
  - How do the numbers relate to performance?

Assignment 4 - Deliverables

The deliverables for the assignment are:

- Well commented source code for a CSP solver based on backtracking search and AC-3, that is able to solve Sudoku boards.
- Your program's solution for each of the four boards in the assignment.
- The number of times your BACKTRACK function was called, and the number of times your BACKTRACK function returned *failure* for each of the four boards. Comment briefly on these numbers for each of the four boards. E.g.:
  - What do the numbers mean?
  - How do the numbers relate to performance?
  - How do they change relative to the difficulty of the boards?

Assignment 4 - Deliverables

The deliverables for the assignment are:

- Well commented source code for a CSP solver based on backtracking search and AC-3, that is able to solve Sudoku boards.
- Your program's solution for each of the four boards in the assignment.
- The number of times your BACKTRACK function was called, and the number of times your BACKTRACK function returned *failure* for each of the four boards. Comment briefly on these numbers for each of the four boards. E.g.:
  - What do the numbers mean?
  - How do the numbers relate to performance?
  - How do they change relative to the difficulty of the boards?
  - Can you make any improvements?

Assignment 4 - Tips

Assignment 4 - Tips

Use the map coloring CSP to debug your code.

Assignment 4 - Tips

Use the map coloring CSP to debug your code.

- It is a smaller problem so it's easier to debug.

Assignment 4 - Tips

Use the map coloring CSP to debug your code.

- It is a smaller problem so it's easier to debug.

Use provided print_sudoku_solution() function to print the board.