Introduction to Artificial Intelligence

Constraint Satisfaction Problems

Andres Mendez-Vazquez

February 5, 2019

Outline

Outline

Introduction

Search Constraint

- A search constraint is a restriction on the set of possible solutions to a search problem.

Introduction

Search Constraint

- A search constraint is a restriction on the set of possible solutions to a search problem.

Examples

- For goal constraints (the standard setting in state space search), we specify goal states, and these incorporate constraints on the goal.
  - Constraints refer to the end of solution paths - the constraints applied to terminal states.
- For path constraints, constraints refer to the path as a whole.
  - Expressed in temporal logic

Introduction

Search Constraint

- A search constraint is a restriction on the set of possible solutions to a search problem.

Examples

- For goal constraints (the standard setting in state space search), we specify goal states, and these incorporate constraints on the goal.
  - Constraints refer to the end of solution paths - the constraints applied to terminal states.
- For path constraints, constraints refer to the path as a whole.
  - Expressed in temporal logic

Introduction

Search Constraint

- A search constraint is a restriction on the set of possible solutions to a search problem.

Examples

- For goal constraints (the standard setting in state space search), we specify goal states, and these incorporate constraints on the goal.
  - Constraints refer to the end of solution paths - the constraints applied to terminal states.
- For path constraints, constraints refer to the path as a whole.
  - Expressed in temporal logic

Outline

Value

Definition

- A value is something that can be assigned to a variable.
- Generally, we reference by v_i the value of the variable X_i.

Value

Definition

- A value is something that can be assigned to a variable.
- Generally, we reference by v_i the value of the variable X_i.

Something Notable

- The nature of these values is a typification of the variables of the problem.

- Symbolic Variables - Colors in a graph.
- etc

Value

Definition

- A value is something that can be assigned to a variable.
- Generally, we reference by v_i the value of the variable X_i.

Something Notable

- The nature of these values is a typification of the variables of the problem.

Examples

- Boolean Variables.
- Symbolic Variables: Colors in a graph.
- etc.

Value

Definition

- A value is something that can be assigned to a variable.
- Generally, we reference by v_i the value of the variable X_i.

Something Notable

- The nature of these values is a typification of the variables of the problem.

Examples

- Boolean Variables.
- Symbolic Variables: Colors in a graph.
- etc.

Value

Definition

- A value is something that can be assigned to a variable.
- Generally, we reference by v_i the value of the variable X_i.

Something Notable

- The nature of these values is a typification of the variables of the problem.

Examples

- Boolean Variables.
- Symbolic Variables: Colors in a graph.
- etc.

Value

Definition

- A value is something that can be assigned to a variable.
- Generally, we reference by v_i the value of the variable X_i.

Something Notable

- The nature of these values is a typification of the variables of the problem.

Examples

- Boolean Variables.
- Symbolic Variables: Colors in a graph.
- etc.

Domain of a Variable

Definition

- The domain of a variable is the set of all the values that this variable can take.

Thus

- If the variable is denoted by X_i , then the most general notation of the domain associated with this variable is either D_i or D_{xi}.

Domain of a Variable

Definition

- The domain of a variable is the set of all the values that this variable can take.

Thus

- If the variable is denoted by X_i , then the most general notation of the domain associated with this variable is either D_i or D_{xi}.

Degree of a Variable

Definition

- The degree of a variable is the number of constraints in which it is involved.

Example

$$X_1 + X_3 + 3X_2 < 15$$
$$7X_2 \times 4X_5 = 84$$
$$2X_1 + 6X_4 - X_2 \geq 9X_3$$

Then, Degree(X_1) = 2, Degree(X_2) = 3, etc

Degree of a Variable

Definition

- The degree of a variable is the number of constraints in which it is involved.

Example

$$X_1 + X_3 + 3X_2 < 15$$
$$7X_2 \times 4X_5 = 84$$
$$2X_1 + 6X_4 - X_2 \geq 9X_3$$

Then, Degree$(X_1) = 2$, Degree$(X_2) = 3$, etc

Constraint

Definition

- A constraint on a set of variables is a restriction on the set of values that these variables can take simultaneously.

Arity of a Constraint

Defintion

The arity of a constraint C is the number of variables involved in C.

Arity of a Constraint

Defintion

The arity of a constraint C is the number of variables involved in C.

Thus

- A constraint is called unary if it relates to a single variable.
- If its arity is equal to two, then we speak of a binary constraint.
- A constraint is called n-ary if its arity is equal to n.

Arity of a Constraint

Defintion

The arity of a constraint C is the number of variables involved in C.

Thus

- A constraint is called unary if it relates to a single variable.
- If its arity is equal to two, then we speak of a binary constraint.
- A constraint is called n-ary if its arity is equal to n.

Arity of a Constraint

Defintion

The arity of a constraint C is the number of variables involved in C.

Thus

- A constraint is called unary if it relates to a single variable.
- If its arity is equal to two, then we speak of a binary constraint.
- A constraint is called n-ary if its arity is equal to n.

Instantiation

Definition

An instantiation I is the simultaneous assignment of values to a set of variables.

Instantiation

Definition

An instantiation I is the simultaneous assignment of values to a set of variables.

Therefore

This instantiation may be in the form of a set of values where each value relates to a variable.

- It can be total or partial.
- They are consistent if the assignement satisfies all the constraints concerned by the variables that it involves.

Instantiation

Definition

An instantiation I is the simultaneous assignment of values to a set of variables.

Therefore

This instantiation may be in the form of a set of values where each value relates to a variable.

- It can be total or partial.
- They are consistent if the assignement satisfies all the constraints concerned by the variables that it involves.

Example

Tuple of values $(v_1, v_2, ..., v_n)$ is a possible instantiation of the variables $(X_1, X_2, ..., X_n)$.

Instantiation

Definition

An instantiation I is the simultaneous assignment of values to a set of variables.

Therefore

This instantiation may be in the form of a set of values where each value relates to a variable.

- It can be total or partial.
- They are consistent if the assignement satisfies all the constraints concerned by the variables that it involves.

Example

Tuple of values $(v_1, v_2, ..., v_n)$ is a possible instantiation of the variables $(X_1, X_2, ..., X_n)$.

Instantiation

Definition

An instantiation I is the simultaneous assignment of values to a set of variables.

Therefore

This instantiation may be in the form of a set of values where each value relates to a variable.

- It can be total or partial.
- They are consistent if the assignement satisfies all the constraints concerned by the variables that it involves.

Example

Tuple of values $(v_1, v_2, ..., v_n)$ is a possible instantiation of the variables $(X_1, X_2, ..., X_n)$.

Outline

Constrain Satisfaction

Where is it used?

- Constraint satisfaction is used to model and solve combinatorial problems.

Something Notable

- Constraint satisfaction relies on a declarative problem description that consists of a set of variables together with their respective domains.

Examples

- $0 \leq X \leq 9$
- $X + Y = 7$
- $X - Y = 5$

Constrain Satisfaction

Where is it used?

- Constraint satisfaction is used to model and solve combinatorial problems.

Something Notable

- Constraint satisfaction relies on a declarative problem description that consists of a set of variables together with their respective domains.

Examples

- $0 \leq X \leq 9$
- $X + Y = 7$
- $X - Y = 5$

Constrain Satisfaction

Where is it used?

- Constraint satisfaction is used to model and solve combinatorial problems.

Something Notable

- Constraint satisfaction relies on a declarative problem description that consists of a set of variables together with their respective domains.

Examples

- $0 \leq X \leq 9$
- $X + Y = 7$
- $X - Y = 5$

A little of NP-Hard

Something Notable

- In constraint solving practice, elementary calculus is often not sufficient to determine the set of feasible solutions.

It is more

- In fact, most constraint satisfaction domains are NP-hard.

A little of NP-Hard

Something Notable

- In constraint solving practice, elementary calculus is often not sufficient to determine the set of feasible solutions.

It is more

- In fact, most constraint satisfaction domains are NP-hard.

Outline

Definition

Constraint Satisfaction Problem

A constraint satisfaction problem (CSP) consists of

- A finite set of variables $V_1, ..., V_n$ over finite domains $D_{v_1}, ..., D_{v_n}$
- A finite set of constraints $C = \{C_1, ..., C_m\}$
  - They are relation between arbitrary variables
- A set $R = \{R_1, ..., R_m\}$ of m relations associated with the constraints where each one of the relations

$$R_i \in D_{i1} \times ... \times D_{ik}$$

  define all combinations of values permitted by C_i.

Definition

Constraint Satisfaction Problem

A constraint satisfaction problem (CSP) consists of

- A finite set of variables $V_1, ..., V_n$ over finite domains $D_{v_1}, ..., D_{v_n}$
- A finite set of constraints $C = \{C_1, ..., C_m\}$
  - They are relation between arbitrary variables
- A set $R = \{R_1, ..., R_m\}$ of m relations associated with the constraints where each one of the relations

$$R_i \in D_{i1} \times ... \times D_{ik}$$

define all combinations of values permitted by C_i.

Definition

Constraint Satisfaction Problem

A constraint satisfaction problem (CSP) consists of

- A finite set of variables $V_1, ..., V_n$ over finite domains $D_{v_1}, ..., D_{v_n}$
- A finite set of constraints $C = \{C_1, ..., C_m\}$
  - They are relation between arbitrary variables
- A set $R = \{R_1, ..., R_m\}$ of m relations associated with the constraints where each one of the relations

$$R_i \in D_{i1} \times ... \times D_{ik}$$

define all combinations of values permitted by C_i.

Definition

Constraint Satisfaction Problem

A constraint satisfaction problem (CSP) consists of

- A finite set of variables $V_1, ..., V_n$ over finite domains $D_{v_1}, ..., D_{v_n}$
- A finite set of constraints $C = \{C_1, ..., C_m\}$
  - They are relation between arbitrary variables
- A set $R = \{R_1, ..., R_m\}$ of m relations associated with the constraints where each one of the relations

$$R_i \in D_{i1} \times ... \times D_{ik}$$

define all combinations of values permitted by C_i.

Definition

Constraint Satisfaction Problem

A constraint satisfaction problem (CSP) consists of

- A finite set of variables $V_1, ..., V_n$ over finite domains $D_{v_1}, ..., D_{v_n}$
- A finite set of constraints $C = \{C_1, ..., C_m\}$
  - They are relation between arbitrary variables
- A set $R = \{R_1, ..., R_m\}$ of m relations associated with the constraints where each one of the relations

$$R_i \in D_{i1} \times ... \times D_{ik}$$

define all combinations of values permitted by C_i.

Definition

Constraint Satisfaction Problem

A constraint satisfaction problem (CSP) consists of

- A finite set of variables $V_1, ..., V_n$ over finite domains $D_{v_1}, ..., D_{v_n}$
- A finite set of constraints $C = \{C_1, ..., C_m\}$
  - They are relation between arbitrary variables
- A set $R = \{R_1, ..., R_m\}$ of m relations associated with the constraints where each one of the relations

$$R_i \in D_{i1} \times ... \times D_{ik}$$

  define all combinations of values permitted by C_i.

Outline

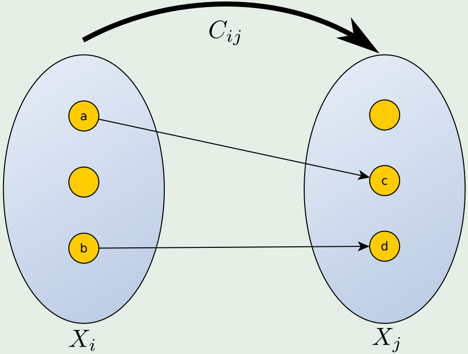
The Different Representations of a CSP

Graphical Representation

- The Most Common...
- Thus, we have two main representations

The Different Representations of a CSP

A representation via a graph local to the constraint



The Different Representations of a CSP

Or a Global Representation

- Via a graph of al CSP constraints.

Associating

- Any CSP (X, D, C, R) a graph of constraints $G = (X, C)$whose nodes represent the variables and the edges the constraints.

The Different Representations of a CSP

Or a Global Representation

- Via a graph of al CSP constraints.

Associating

- Any CSP (X, D, C, R) a graph of constraints $G = (X, C)$whose nodes represent the variables and the edges the constraints.

We not only have these representations

Representation in Extensions

- The set of pairs authorized for the binary constraints or more generally the n-uplets authorized for the n-ary constraints.

Representation in Intention

- The constraints are in the form of equations or predicates.

We not only have these representations

Representation in **Extensions**

- The set of pairs authorized for the binary constraints or more generally the n-uplets authorized for the n-ary constraints.

Representation in **Intention**

- The constraints are in the form of equations or predicates.

Nevertheless

Finally, we want a solution

- It is a complete assignment of values to variables satisfying all the constraints.

IMPORTANT

For the sake of simplicity

- We are ruling out continuous variables in the definition!!!

Outline

Example

We have

- Binary CSP is a CSP where the constraints involve only two variables

For example take the following CSP

- $X + Y = Z,\ X < Y$
- Domain $D_X = \{1, 2\}$, $D_Y = \{3, 4\}$ and $D_Z = \{5, 6\}$

Example

We have

- Binary CSP is a CSP where the constraints involve only two variables

For example take the following CSP

- $X + Y = Z, \; X < Y$
- Domain $D_X = \{1, 2\}$, $D_Y = \{3, 4\}$ and $D_Z = \{5, 6\}$

Classic Example

Task

- To place eight queens on a chess board, but with at most one queen in the same row, column, or diagonal.

Classic Example

Task

- To place eight queens on a chess board, but with at most one queen in the same row, column, or diagonal.

Variables

- X_i denotes the row of the queen i, $i \in \{1, ..., 8\}$.
- Y_i denotes the column of the queen i, $i \in \{1, ..., 8\}$.

Classic Example

Task

- To place eight queens on a chess board, but with at most one queen in the same row, column, or diagonal.

Variables

- X_i denotes the row of the queen i, $i \in \{1, ..., 8\}$.
- Y_i denotes the column of the queen i, $i \in \{1, ..., 8\}$.

Domain

- $D_{X_1} = ... = D_{X_8} = \{1, ..., 8\}$.
- $D_{Y_1} = ... = D_{Y_8} = \{1, ..., 8\}$.

Classic Example

Task

- To place eight queens on a chess board, but with at most one queen in the same row, column, or diagonal.

Variables

- X_i denotes the row of the queen i, $i \in \{1, ..., 8\}$.
- Y_i denotes the column of the queen i, $i \in \{1, ..., 8\}$.

Domain

- $D_{X_1} = ... = D_{X_8} = \{1, ..., 8\}$.
- $D_{Y_1} = ... = D_{Y_8} = \{1, ..., 8\}$.

Classic Example

Task

- To place eight queens on a chess board, but with at most one queen in the same row, column, or diagonal.

Variables

- X_i denotes the row of the queen i, $i \in \{1, ..., 8\}$.
- Y_i denotes the column of the queen i, $i \in \{1, ..., 8\}$.

Domain

- $D_{X_1} = ... = D_{X_8} = \{1, ..., 8\}$.
- $D_{Y_1} = ... = D_{Y_8} = \{1, ..., 8\}$.

The Final Constraints

Constraints

The constraints that induce no conflict are

- $X_i \neq X_j$ (no vertical threat) for all $1 \leq i \neq j \leq 8$
- $Y_i \neq Y_j$ (no horizontal threat) for all $1 \leq i \neq j \leq 8$
- $|X_i - X_j| \neq |Y_i - Y_j|$ (no diagonal threat) for all $1 \leq i \neq j \leq 8$

The Final Constraints

Constraints

The constraints that induce no conflict are

- $X_i \neq X_j$ (no vertical threat) for all $1 \leq i \neq j \leq 8$
- $Y_i \neq Y_j$ (no horizontal threat) for all $1 \leq i \neq j \leq 8$
- $|X_i - X_j| \neq |Y_i - Y_j|$ (no diagonal threat) for all $1 \leq i \neq j \leq 8$

The Final Constraints

Constraints

The constraints that induce no conflict are

- $X_i \neq X_j$ (no vertical threat) for all $1 \leq i \neq j \leq 8$
- $Y_i \neq Y_j$ (no horizontal threat) for all $1 \leq i \neq j \leq 8$
- $|X_i - X_j| \neq |Y_i - Y_j|$ (no diagonal threat) for all $1 \leq i \neq j \leq 8$

The Final Constraints

Constraints

The constraints that induce no conflict are

- $X_i \neq X_j$ (no vertical threat) for all $1 \leq i \neq j \leq 8$
- $Y_i \neq Y_j$ (no horizontal threat) for all $1 \leq i \neq j \leq 8$
- $|X_i - X_j| \neq |Y_i - Y_j|$ (no diagonal threat) for all $1 \leq i \neq j \leq 8$

The map-coloring problem

Definition

You need to color a map with k colors in such a way that the two neighboring areas, having a common border, are not of the same color.

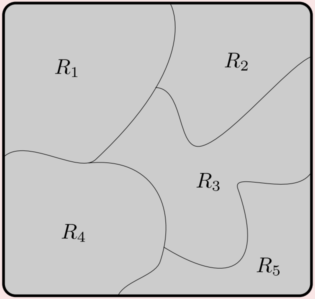
Example

The map-coloring problem

Definition

You need to color a map with k colors in such a way that the two neighboring areas, having a common border, are not of the same color.

Example



The CSP for the Map-Coloring

You have the triplet (X, D, C)

- $X = \{R_1, R_2, R_3, R_4, R_5\}$.
- $D = \{D_1, D_2, D_3, D_4, D_5\}$ where $D_i = \{r, g, b\}$

The CSP for the Map-Coloring

You have the triplet (X, D, C)

- $X = \{R_1, R_2, R_3, R_4, R_5\}$.
- $D = \{D_1, D_2, D_3, D_4, D_5\}$ where $D_i = \{r, g, b\}$

With the following Constraints

$$C = \{R_1 \neq R_2, R_1 \neq R_3, R_1 \neq R_4, R_3 \neq R_4$$
$$R_2 \neq R_3, R_3 \neq R_5, R_4 \neq R_5\}$$

The CSP for the Map-Coloring

You have the triplet (X, D, C)

- $X = \{R_1, R_2, R_3, R_4, R_5\}$.
- $D = \{D_1, D_2, D_3, D_4, D_5\}$ where $D_i = \{r, g, b\}$

With the following Constraints

$$
\begin{gathered}
C = \{R_1 \neq R_2, R_1 \neq R_3, R_1 \neq R_4, R_3 \neq R_4 \\
R_2 \neq R_3, R_3 \neq R_5, R_4 \neq R_5\}
\end{gathered}
$$

Outline

How do we solve this?

Using an efficient Algorithm

Such a problem formulation calls for an efficient search algorithm to find a feasible variable assignment representing valid placements of the queens on the board.

Naive solution

A naive strategy considers all 8^8 possible assignments, which can easily be reduced to $8!$.

Better

We need a refined approach maintains a vector for a partial assignment in a vector, which grows with increasing depth and shrinks with each backtrack.

How do we solve this?

Using an efficient Algorithm

Such a problem formulation calls for an efficient search algorithm to find a feasible variable assignment representing valid placements of the queens on the board.

Naive solution

A naive strategy considers all 8^8 possible assignments, which can easily be reduced to 8!.

Better

We need a refined approach maintains a vector for a partial assignment in a vector, which grows with increasing depth and shrinks with each backtrack.

How do we solve this?

Using an efficient Algorithm

Such a problem formulation calls for an efficient search algorithm to find a feasible variable assignment representing valid placements of the queens on the board.

Naive solution

A naive strategy considers all 8^8 possible assignments, which can easily be reduced to 8!.

Better

We need a refined approach maintains a vector for a partial assignment in a vector, which grows with increasing depth and shrinks with each backtrack.

How do we solve this?

In addition

To limit the branching during the search, we additionally maintain a global data structure to mark all places that are in conflict with the current assignment.

This is known as consistency

Once, we can define this, it is possible to talk of feasible algorithms!!!

How do we solve this?

In addition

To limit the branching during the search, we additionally maintain a global data structure to mark all places that are in conflict with the current assignment.

This is known as consistency

Once, we can define this, it is possible to talk of feasible algorithms!!!

Outline

Consistency

What is this?

Consistency is an inference mechanism to rule out certain variable assignments, which in turn enhances the search.

Simple version

The simplest consistency check tests a current assignment against the set of constraints.

Consistency

What is this?

Consistency is an inference mechanism to rule out certain variable assignments, which in turn enhances the search.

Simple version

The simplest consistency check tests a current assignment against the set of constraints.

Simple Algorithm

Procedure Consistent

Input: Label set L, constraints C

Output: L satisfies C true/false

1. **for each** $c \in C$
2. **if** $Variables(c) \subseteq L$
3. **if not** $Satisfied(c, L)$
4. **return false** // *When an inconsistency happens*
5. **return true**

Simple Algorithm

Procedure Consistent

Input: Label set L, constraints C

Output: L satisfies C true/false

1. **for each** $c \in C$
2.   **if** $Variables\,(c) \subseteq L$
3.   **if not** $Satisfied\,(c, L)$
4.   **return false** // *When an inconsistency happens*
5. **return true**

With

- $Variables\,(c)$ denotes the set of variables in mentioned in the constraint c.
- $Satisfied(c, L)$ to denote if the constraint c is satisfied by the current label set L(values to variables).

However

We need something better

There is a long list of algorithms for this.

However

We will look to algorithms that check between constraints between two variables.

However

We need something better

There is a long list of algorithms for this.

However

We will look to algorithms that check between constraints between two variables.

Outline

Arc Consistency

Introduction

Arc consistency is one of the most powerful propagation techniques for binary constraints.

What are binary constraints?

A binary constraint is a constraint involving only two variables

Example over Graphs

Arc Consistency

Introduction

Arc consistency is one of the most powerful propagation techniques for binary constraints.

What are binary constraints?

A binary constraint is a constraint involving only two variables

Example over Graphs

Arc Consistency

Introduction

Arc consistency is one of the most powerful propagation techniques for binary constraints.

What are binary constraints?

A binary constraint is a constraint involving only two variables

Example over Graphs

Basic Definitions

Definition - K-Consistency

A CSP (X, D, C, R) is k-consistent if and only if, for any n-tuplet of k variables $(X_1, ..., X_k)$ of X, any consistent $k-1$ instantiation may be extended to a consistent instantiation with the k^{th} variable.

Definition - Strong K-Consistency

A CSP $P(X, D, C, R)$ is said to be strongly k-consistent if and only if, $\forall i, 1 \leq i \leq k$, P is i-consistent.

Basic Definitions

Definition - K-Consistency

A CSP (X, D, C, R) is k-consistent if and only if, for any n-tuplet of k variables $(X_1, ..., X_k)$ of X, any consistent $k-1$ instantiation may be extended to a consistent instantiation with the k^{th} variable.

Definition - Strong K-Consistency

A CSP $P(X, D, C, R)$ is said to be strongly k-consistent if and only if, $\forall i, 1 \leq i \leq k$, P is i-consistent.

Basic Definitions

Definition - Node Consistency

A node-consistent CSP (X, D, C, R) is a 1-consistent CSP. This consistency is only verified if for any X_i variable of X, and for any v_i value of D_i, the partial assignment (X_i, v_i) satisfies all the unary constraints of C involving this variable.

Arc Consistency

Definition

A CSP (X, D, C, R) is called ***arc consistent*** if and only if, for any couple of variables (X_i, X_j) of X, each couple represents an arc in the associated constraint graph, and for any value v_i from the domain D_i that satisfies the unary constraints involving X_i, there is a value v_j in the domain D_j compatible with v_i.

Something Notable

- Initially presented by A.K. Mackworth.

Arc Consistency

Definition

A CSP (X, D, C, R) is called ***arc consistent*** if and only if, for any couple of variables (X_i, X_j) of X, each couple represents an arc in the associated constraint graph, and for any value v_i from the domain D_i that satisfies the unary constraints involving X_i, there is a value v_j in the domain D_j compatible with v_i.

Something Notable

- Initially presented by A.K. Mackworth.
- Arc consistency is expressed on each couple of variables of a problem with binary constraints.
  - It is equivalent to the 2-consistency.

Arc Consistency

Definition

A CSP (X, D, C, R) is called ***arc consistent*** if and only if, for any couple of variables (X_i, X_j) of X, each couple represents an arc in the associated constraint graph, and for any value v_i from the domain D_i that satisfies the unary constraints involving X_i, there is a value v_j in the domain D_j compatible with v_i.

Something Notable

- Initially presented by A.K. Mackworth.
- Arc consistency is expressed on each couple of variables of a problem with binary constraints.
  - It is equivalent to the 2-consistency.

Example

Consider a simple CSP with variables A and B

- Domains $D_A = \{1, 2\}$ and $D_B = \{1, 2, 3\}$
- With constraint $A < B$

Thus...

We can easily remove 1 from D_B.

Example

Consider a simple CSP with variables A and B

- Domains $D_A = \{1, 2\}$ and $D_B = \{1, 2, 3\}$
- With constraint $A < B$

Thus...

We can easily remove 1 from D_B.

What kind of problems?

To solve this kind of problems

Graph coloring problem:

- Given a planar graph, assign one of 4 colors to each vertex such that any two adjacent vertices have different colors.

Example

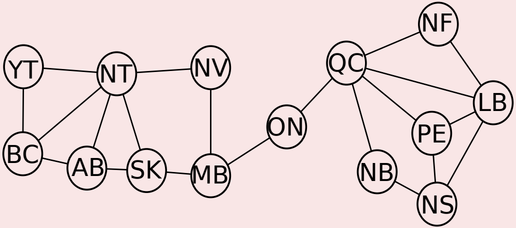
What kind of problems?

To solve this kind of problems

Graph coloring problem:

- Given a planar graph, assign one of 4 colors to each vertex such that any two adjacent vertices have different colors.

Example



Outline

Two Main Algorithms

We will look at

- AC-1
- AC-3
  - They both rely in a very simple function called Revise!!!

Two Main Algorithms

We will look at

- AC-1
- AC-3
  - They both rely in a very simple function called Revise!!!

It is applied to

Couple of variables (X_i, X_j) connected by a constraint C_{ij} by removing the locally inconsistent values from the X_i.

- The arc consistency is verified if and only if all the arcs on the graph of constraints are arc consistent.

Two Main Algorithms

We will look at

- AC-1
- AC-3
  - They both rely in a very simple function called Revise!!!

It is applied to

Couple of variables (X_i, X_j) connected by a constraint C_{ij} by removing the locally inconsistent values from the X_i.

Where

- This couple of variables represents an arc in the graph of constraints often denoted by (i, j).
- The arc consistency is verified if and only if all the arcs on the graph of constraints are arc consistent.

Two Main Algorithms

We will look at

- AC-1
- AC-3
  - They both rely in a very simple function called Revise!!!

It is applied to

Couple of variables (X_i, X_j) connected by a constraint C_{ij} by removing the locally inconsistent values from the X_i.

Where

- This couple of variables represents an arc in the graph of constraints often denoted by (i, j).
- The arc consistency is verified if and only if all the arcs on the graph of constraints are arc consistent.

Two Main Algorithms

We will look at

- AC-1
- AC-3
    - They both rely in a very simple function called Revise!!!

It is applied to

Couple of variables (X_i, X_j) connected by a constraint C_{ij} by removing the locally inconsistent values from the X_i.

Where

- This couple of variables represents an arc in the graph of constraints often denoted by (i, j).
- The arc consistency is verified if and only if all the arcs on the graph of constraints are arc consistent.

Two Main Algorithms

We will look at

- AC-1
- AC-3
  - They both rely in a very simple function called Revise!!!

It is applied to

Couple of variables (X_i, X_j) connected by a constraint C_{ij} by removing the locally inconsistent values from the X_i.

Where

- This couple of variables represents an arc in the graph of constraints often denoted by (i, j).
- The arc consistency is verified if and only if all the arcs on the graph of constraints are arc consistent.

Revise Procedure

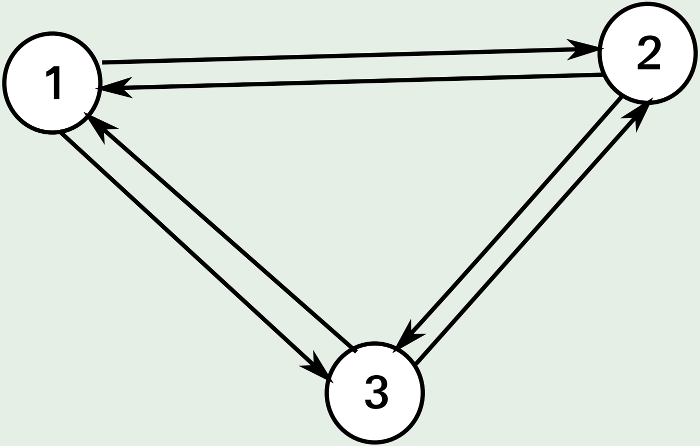
Revise(i, j)

output Boolean

1. $CHANGE = False$
2. **for each** $x \in D_i$
3. **If there is no** $y \in D_j$ **such that** $R_{ij}(x, y)$ **is true then**
4. **delete** x **from** D_i
5. $CHANGE = True$
6. **return** $CHANGE$

Example we are going to use

Graph to color



Outline

AC-1

AC-1

- Early version is due to ROSENFELD A., HUMMEL R., ZUCKER S.

- (X_i, X_j) are linked by a constraint C_{ij}
- The algorithm visits each couple (X_i, X_j) and removes all the values that violate C_{ij} from domain D_i

AC-1

AC-1

- Early version is due to ROSENFELD A., HUMMEL R., ZUCKER S.

What is this?

- The main mechanism for implementing this procedure is based on a list Q supplied by all the couples of variables (X_i, X_j)
- (X_j, X_i) are linked by a constraint C_{ij}.
- The algorithm visits each couple (X_i, X_j) and removes all the values that violate C_{ij} from domain D_i.

AC-1

AC-1

- Early version is due to ROSENFELD A., HUMMEL R., ZUCKER S.

What is this?

- The main mechanism for implementing this procedure is based on a list Q supplied by all the couples of variables (X_i, X_j)
- (X_j, X_i) are linked by a constraint C_{ij}.
- The algorithm visits each couple (X_i, X_j) and removes all the values that violate C_{ij} from domain D_i.

AC-1

AC-1

- Early version is due to ROSENFELD A., HUMMEL R., ZUCKER S.

What is this?

- The main mechanism for implementing this procedure is based on a list Q supplied by all the couples of variables (X_i, X_j)
- (X_j, X_i) are linked by a constraint C_{ij}.
- The algorithm visits each couple (X_i, X_j) and removes all the values that violate C_{ij} from domain D_i.

AC-1 Algorithms

Procedure AC1

1. $Q = \{(i,j) | C_{ij} \in C, i \neq j\}$.
2. **Repeat**
3. $\quad CHANGE = False$
4. $\quad$ **for each** $(i,j) \in Q$ **do**
5. $\quad\quad CHANGE = (Revise(i,j) \vee CHANGE)$
6. **Until** $\neg Change$

Example

We have that

- In the first column, the current couple of variables (i, j) being treated by the revise procedure is colored in **red**.
- In the second column, the value (color) removed by the revise procedure is colored in **red**.
- In the second column, the final domains obtained after performing the whole AC-1 are emphasized.

Example

We have that

- In the first column, the current couple of variables (i, j) being treated by the revise procedure is colored in **red**.
- In the second column, the value (color) removed by the revise procedure is colored in **red**.
- In the second column, the final domains obtained after performing the whole AC-1 are emphasized.

Example

We have that

- In the first column, the current couple of variables (i, j) being treated by the revise procedure is colored in **red**.
- In the second column, the value (color) removed by the revise procedure is colored in **red**.
- In the second column, the final domains obtained after performing the whole AC-1 are emphasized.

Example

Then

ITERATION	Q	D_i	Change
1(Repeat).1(For)	$\{(1,2);(2,1);(1,3)$ $(3,1);(2,3);(3,2)\}$	$D_1 = \{R,G,B\}$ $D_2 = \{R,G\}$ $D_3 = \{G\}$	FALSE
1.2	$\{(1,2);(2,1);(1,3)$ $(3,1);(2,3);(3,2)\}$	$D_1 = \{R,G,B\}$ $D_2 = \{R,G\}$ $D_3 = \{G\}$	FALSE
1.3	$\{(1,2);(2,1);(1,3)$ $(3,1);(2,3);(3,2)\}$	$D_1 = \{R,G,B\}$ $D_2 = \{R,G\}$ $D_3 = \{G\}$	TRUE

Example

Then

ITERATION	Q	D_i	Change
1(Repeat).1(For)	$\{(1,2);(2,1);(1,3)$ $(3,1);(2,3);(3,2)\}$	$D_1 = \{R,G,B\}$ $D_2 = \{R,G\}$ $D_3 = \{G\}$	FALSE
1.2	$\{(1,2);(2,1);(1,3)$ $(3,1);(2,3);(3,2)\}$	$D_1 = \{R,G,B\}$ $D_2 = \{R,G\}$ $D_3 = \{G\}$	FALSE
1.3	$\{(1,2);(2,1);(1,3)$ $(3,1);(2,3);(3,2)\}$	$D_1 = \{R,G,B\}$ $D_2 = \{R,G\}$ $D_3 = \{G\}$	TRUE

Example

Then

ITERATION	Q	D_i	Change
1(Repeat).1(For)	$\{(1,2);(2,1);(1,3)$ $(3,1);(2,3);(3,2)\}$	$D_1=\{R,G,B\}$ $D_2=\{R,G\}$ $D_3=\{G\}$	FALSE
1.2	$\{(1,2);(2,1);(1,3)$ $(3,1);(2,3);(3,2)\}$	$D_1=\{R,G,B\}$ $D_2=\{R,G\}$ $D_3=\{G\}$	FALSE
1.3	$\{(1,2);(2,1);(1,3)$ $(3,1);(2,3);(3,2)\}$	$D_1=\{R,G,B\}$ $D_2=\{R,G\}$ $D_3=\{G\}$	TRUE

Example

Then

ITERATION	Q	D_i	Change
1.4	$\{(1,2)\,;(2,1);(1,3)$ $(3,1);(2,3)\,;(3,2)\}$	$D_1=\{R,B\}$ $D_2=\{R,G\}$ $D_3=\{G\}$	TRUE
1.5	$\{(1,2)\,;(2,1);(1,3)$ $(3,1)\,;(2,3);(3,2)\}$	$D_1=\{R,B\}$ $D_2=\{R,G\}$ $D_3=\{G\}$	TRUE
1.6	$\{(1,2)\,;(2,1);(1,3)$ $(3,1)\,;(2,3)\,;(3,2)\}$	$D_1=\{R,B\}$ $D_2=\{R\}$ $D_3=\{G\}$	TRUE

Example

Then

ITERATION	Q	D_i	Change
1.4	$\{(1,2)\,;(2,1);(1,3)$ $(3,1);(2,3)\,;(3,2)\}$	$D_1=\{R,B\}$ $D_2=\{R,G\}$ $D_3=\{G\}$	TRUE
1.5	$\{(1,2)\,;(2,1);(1,3)$ $(3,1)\,;(2,3);(3,2)\}$	$D_1=\{R,B\}$ $D_2=\{R,G\}$ $D_3=\{G\}$	TRUE
1.6	$\{(1,2)\,;(2,1);(1,3)$ $(3,1)\,;(2,3)\,;(3,2)\}$	$D_1=\{R,B\}$ $D_2=\{R\}$ $D_3=\{G\}$	TRUE

Example

Then

ITERATION	Q	D_i	Change
1.4	$\{(1,2)\,;(2,1);(1,3)$ $(3,1);(2,3)\,;(3,2)\}$	$D_1 = \{R,B\}$ $D_2 = \{R,G\}$ $D_3 = \{G\}$	TRUE
1.5	$\{(1,2)\,;(2,1);(1,3)$ $(3,1)\,;(2,3);(3,2)\}$	$D_1 = \{R,B\}$ $D_2 = \{R,G\}$ $D_3 = \{G\}$	TRUE
1.6	$\{(1,2)\,;(2,1);(1,3)$ $(3,1)\,;(2,3)\,;(3,2)\}$	$D_1 = \{R,B\}$ $D_2 = \{R\}$ $D_3 = \{G\}$	TRUE

Example

Then

ITERATION	Q	D_i	Change
2.1	$\{(1,2);(2,1);(1,3)$ $(3,1);(2,3);(3,2)\}$	$D_1=\{R,B\}$ $D_2=\{R\}$ $D_3=\{G\}$	TRUE
2.2	$\{(1,2);(2,1);(1,3)$ $(3,1);(2,3);(3,2)\}$	$D_1=\{B\}$ $D_2=\{R\}$ $D_3=\{G\}$	TRUE

Example

Then

ITERATION	Q	D_i	Change
2.1	$\{(1,2);(2,1);(1,3)$ $(3,1);(2,3);(3,2)\}$	$D_1 = \{R, B\}$ $D_2 = \{R\}$ $D_3 = \{G\}$	TRUE
2.2	$\{(1,2);(2,1);(1,3)$ $(3,1);(2,3);(3,2)\}$	$D_1 = \{B\}$ $D_2 = \{R\}$ $D_3 = \{G\}$	TRUE

Complexity of AC-1

Something Notable

The systematic nature of the revisions of all the problem's arcs, each time a value is removed, deteriorates the AC-1 performances, since the removal of a value from a variable X.

Complexity of AC-1

Something Notable

The systematic nature of the revisions of all the problem's arcs, each time a value is removed, deteriorates the AC-1 performances, since the removal of a value from a variable X.

With n variables, we have that

- $d = \max \{|D_i|\}_{i=1}^{n}$
- The worst case on the number of constraints $\frac{n(n-1)}{2}$

Complexity of AC-1

Something Notable

The systematic nature of the revisions of all the problem's arcs, each time a value is removed, deteriorates the AC-1 performances, since the removal of a value from a variable X.

With n variables, we have that

- $d = \max \{|D_i|\}_{i=1}^{n}$
- The worst case on the number of constraints $\frac{n(n-1)}{2}$

We have that

- Temporal Complexity: $O\left(n^3 d^3\right)$
- Spatial Complexity: $O\left(n^2\right)$

Complexity of AC-1

Something Notable

The systematic nature of the revisions of all the problem's arcs, each time a value is removed, deteriorates the AC-1 performances, since the removal of a value from a variable X.

With n variables, we have that

- $d = \max\left\{|D_i|\right\}_{i=1}^{n}$
- The worst case on the number of constraints $\frac{n(n-1)}{2}$

We have that

- Temporal Complexity: $O\left(n^3 d^3\right)$
- Spatial Complexity: $O\left(n^2\right)$

Can we improve the AC-1?

If we observe the idea of locality

- Arc consistency can be obtained by testing the neighboring area that consists of the set of variables connected by a binary constraint.

Can we improve the AC-1?

If we observe the idea of locality

- Arc consistency can be obtained by testing the neighboring area that consists of the set of variables connected by a binary constraint.

We do the following

- The algorithm uses a queue structure Q.
- To determine all non-viable values, AC-3 seeks support for each value on each constraints.

Can we improve the AC-1?

If we observe the idea of locality

- Arc consistency can be obtained by testing the neighboring area that consists of the set of variables connected by a binary constraint.

We do the following

- The algorithm uses a queue structure Q.
- To determine all non-viable values, AC-3 seeks support for each value on each constraints.

Then

Then

At each step a pop element, an arc (i,j), is revised using the **Revise** procedure.

During this revision

If the removal of a value v_i occurs in D_i, the set of arcs (k,i) such as $k \neq i$ and $k \neq j$ are added to Q (If not there already).

Next

AC-3 re-examines the viability of all the values $v_k \in D_k$ in relation to C_{ki}.

Then

Then

At each step a pop element, an arc (i, j), is revised using the **Revise** procedure.

During this revision

If the removal of a value v_i occurs in D_i, the set of arcs (k, i) such as $k \neq i$ and $k \neq j$ are added to Q (If not there already).

Next

AC-3 re-examines the viability of all the values $v_k \in D_k$ in relation to C_{ki}.

Then

Then

At each step a pop element, an arc (i, j), is revised using the **Revise** procedure.

During this revision

If the removal of a value v_i occurs in D_i, the set of arcs (k, i) such as $k \neq i$ and $k \neq j$ are added to Q (If not there already).

Next

AC-3 re-examines the viability of all the values $v_k \in D_k$ in relation to C_{ki}.

Important!!!

We can have the following

It may happen that (i, v_i) is the sole support for certain values of D_k and the removal of v_i makes them not arc consistent.

Outline

Algorithm AC-3

Procedure AC-3

Input: Set of variables V, set of domains D, set of constraints C

Output: Satisfiable true/false, restricted set of domains

1. Q=$\{(i,j)\,|c_{ij} \in C, i \neq j\}$ **// Where Q is a Queue**
2. **while** $(Q \neq \emptyset)$
3. $c = Q.pop()$
4.   **if Revise**(i,j)
5. $Q = Q \cup \{(k,i)\,|c_{ki} \in C, k \neq i, k \neq j\}$

Explanation

Then

1. All the arcs are added to the queue Q.
2. If an arc has been removed... it can be added again to Q **(Line 5)**
3. This allows to again revise that arc so maybe something else to discard.

Explanation

Then

1. All the arcs are added to the queue Q.
2. If an arc has been removed... it can be added again to Q **(Line 5)**
3. This allows to again revise that arc so maybe something else to discard.

Explanation

Then

1. All the arcs are added to the queue Q.
2. If an arc has been removed... it can be added again to Q **(Line 5)**
3. This allows to again revise that arc so maybe something else to discard.

Complexity

Something Notable

This algorithm re-examines the viability of more values than necessary (re-examines all the values even those that are not concerned by the removal).

Complexity

Something Notable

This algorithm re-examines the viability of more values than necessary (re-examines all the values even those that are not concerned by the removal).

Then

- For a complete graph of constraints, the Q will see the insertion of $O\left(n^2 d\right)$ arcs
- Temporal Complexity $O\left(n^2 d^3\right)$ - More efficient than AC-3
- Spatial Complexity $O\left(n^2\right)$

Complexity

Something Notable

This algorithm re-examines the viability of more values than necessary (re-examines all the values even those that are not concerned by the removal).

Then

- For a complete graph of constraints, the Q will see the insertion of $O\left(n^2 d\right)$ arcs
- Temporal Complexity $O\left(n^2 d^3\right)$ - More efficient than AC-3
- Spatial Complexity $O\left(n^2\right)$

Complexity

Something Notable

This algorithm re-examines the viability of more values than necessary (re-examines all the values even those that are not concerned by the removal).

Then

- For a complete graph of constraints, the Q will see the insertion of $O\left(n^2 d\right)$ arcs
- Temporal Complexity $O\left(n^2 d^3\right)$ - More efficient than AC-3
- Spatial Complexity $O\left(n^2\right)$

Example

Then

Iteration	Q	D_i	Revise (i, j)
1	{(1, 2); (2, 1); (1, 3); (3, 1); (2, 3); (3, 2)}	D1 = {R, G, B} D2 = {R, G} D3 = {G}	FALSE
2	{(2, 1); (1, 3); (3, 1); (2, 3); (3, 2)}	D1 = {R, G, B} D2 = {R, G} D3 = {G}	FALSE
3	{(1, 3); (3, 1); (2, 3); (3, 2)}	D1 = { R, G, B} D2 = {R, G} D3 = {G}	TRUE
4	{(3, 1); (2, 3); (3, 2) ∪ **(2, 1)**}	D1 = {R, B} D2 = {R, G} D3 = {G}	FALSE

Example

Then

5	{(2, 3); (3, 2); (2, 1)}	D1 = {R, B} D2 = {R, G} D3 = {G}	TRUE
6	{(3, 2); (2, 1) ∪ **(1, 2)}**	D1 = {R, B} D2 = {R} D3 = {G}	FALSE
7	{(2, 1); (1, 2)}	D1 = {R, B} D2 = {R} D3 = {G}	FALSE
8	{(1, 2)}	D1 = {R, B} D2 = {R} D3 = {G}	TRUE
9	{∅ ∪ **(3, 1)**}	D1 = {B} D2 = {R} D3 = {G}	FALSE

There are other methods for arc consistency

You can look at them in

“Constraint Satisfaction Problems: CSP Formalisms and Techniques” by Khaled Ghedira.

Outline

How can we use this?

Various techniques for solving CSP have been developed

Classification of them:

- Complete methods that guarantee completeness (quality) at the expense of efficiency (temporal complexity)
- Incomplete methods that sacrifice completeness for the sake of efficiency.

How can we use this?

Various techniques for solving CSP have been developed

Classification of them:

- Complete methods that guarantee completeness (quality) at the expense of efficiency (temporal complexity)
- Incomplete methods that sacrifice completeness for the sake of efficiency.

We will look at a complete resolution method: Backtracking

- Remember it?? Solving NP-Problems

- It takes a sequence V of variables of X to be instantiated (initially X including all the variables)
- An initially empty instantiation I as arguments

How can we use this?

Various techniques for solving CSP have been developed

Classification of them:

- Complete methods that guarantee completeness (quality) at the expense of efficiency (temporal complexity)
- Incomplete methods that sacrifice completeness for the sake of efficiency.

We will look at a complete resolution method: Backtracking

- Remember it?? Solving NP-Problems

Properties

- It uses Depth-First Search.
- It takes a sequence V of variables of X to be instantiated (Initially X including all the variables).
- An initially empty instantiation I as arguments.

How can we use this?

Various techniques for solving CSP have been developed

Classification of them:

- Complete methods that guarantee completeness (quality) at the expense of efficiency (temporal complexity)
- Incomplete methods that sacrifice completeness for the sake of efficiency.

We will look at a complete resolution method: Backtracking

- Remember it?? Solving NP-Problems

Properties

- It uses Depth-First Search.
- It takes a sequence V of variables of X to be instantiated (Initially X including all the variables).
- An initially empty instantiation I as arguments.

How can we use this?

Various techniques for solving CSP have been developed

Classification of them:

- Complete methods that guarantee completeness (quality) at the expense of efficiency (temporal complexity)
- Incomplete methods that sacrifice completeness for the sake of efficiency.

We will look at a complete resolution method: Backtracking

- Remember it?? Solving NP-Problems

Properties

- It uses Depth-First Search.
- It takes a sequence V of variables of X to be instantiated (Initially X including all the variables).
- An initially empty instantiation I as arguments.

How can we use this?

Various techniques for solving CSP have been developed

Classification of them:

- Complete methods that guarantee completeness (quality) at the expense of efficiency (temporal complexity)
- Incomplete methods that sacrifice completeness for the sake of efficiency.

We will look at a complete resolution method: Backtracking

- Remember it?? Solving NP-Problems

Properties

- It uses Depth-First Search.
- It takes a sequence V of variables of X to be instantiated (Initially X including all the variables).
- An initially empty instantiation I as arguments.

How can we use this?

Various techniques for solving CSP have been developed

Classification of them:

- Complete methods that guarantee completeness (quality) at the expense of efficiency (temporal complexity)
- Incomplete methods that sacrifice completeness for the sake of efficiency.

We will look at a complete resolution method: Backtracking

- Remember it?? Solving NP-Problems

Properties

- It uses Depth-First Search.
- It takes a sequence V of variables of X to be instantiated (Initially X including all the variables).
- An initially empty instantiation I as arguments.

Backtracking Algorithm

BackTracking(V, I)

```
If V = ∅ then
        I is a solution
else
        Let x ∈ V
        for each v ∈ D_x do
                If I ∪ {(x, v)} is consistent then
                        BackTracking (V − {x} , I ∪ (x, v))
```

Outline

Example

Pruning Example

Given the possible values that you can give to two literals:

x_1	x_2
1	1
1	0
0	1
$\mathbf{0}$	$\mathbf{0}$

It is possible to prune a quarter of the entire search space... Can this be systematically exploited?

An example of exploiting this idea in SAT solvers

Consider the following Boolean formula $\phi(w, x, y, z)$

$$(w \vee x \vee y \vee z) \wedge (w \vee \neg x) \wedge (x \vee \neg y) \wedge (y \vee \neg z) \wedge (z \vee \neg w) \wedge (\neg w \vee \neg z)$$

We start branching in one variable, we can choose w

Note: This selection does not violate any of the clauses of $\phi(w, x, y, z)$

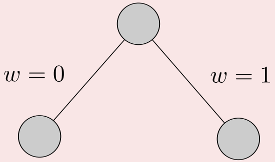
An example of exploiting this idea in SAT solvers

Consider the following Boolean formula $\phi(w, x, y, z)$

$(w \lor x \lor y \lor z) \land (w \lor \neg x) \land (x \lor \neg y) \land (y \lor \neg z) \land (z \lor \neg w) \land (\neg w \lor \neg z)$

We start branching in one variable, we can choose w

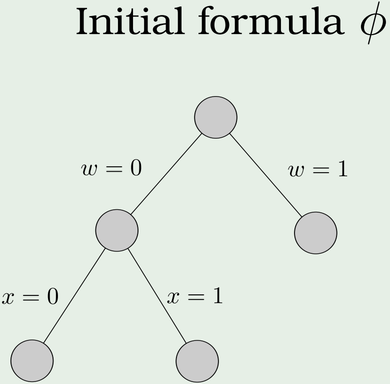
Initial formula ϕ



Note: This selection does not violate any of the clauses of $\phi(w, x, y, z)$

Now

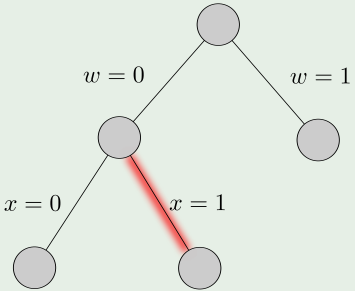
The partial assignment $w = 0, x = 1$ violates the clause $(w \vee \neg x)$



Now

Then, we prune that branch

Initial formula ϕ



In addition

What if $w = 0, x = 0$

instantiation

Then, the following clauses are satisfied

1. $\neg w = 1$
2. $\neg x = 1$

Thus, we have the following left

1. Before
   1. $(w \vee x \vee y \vee z) \wedge (w \vee \neg x) \wedge (x \vee \neg y) \wedge (y \vee \neg z) \wedge (z \vee \neg w) \wedge (\neg w \vee \neg z)$
2. After
   1. $(0 \vee 0 \vee y \vee z) \wedge (0 \vee 1) \wedge (0 \vee \neg y) \wedge (y \vee \neg z) \wedge (z \vee 1) \wedge (1 \vee \neg z)$

In addition

What if $w = 0, x = 0$

instantiation

Then, the following clauses are satisfied

1. $\neg w = 1$
2. $\neg x = 1$

Thus, we have the following left

1. Before
   1. $(w \vee x \vee y \vee z) \wedge (w \vee \neg x) \wedge (x \vee \neg y) \wedge (y \vee \neg z) \wedge (z \vee \neg w) \wedge (\neg w \vee \neg z)$
2. After
   1. $(0 \vee 0 \vee y \vee z) \wedge (0 \vee 1) \wedge (0 \vee \neg y) \wedge (y \vee \neg z) \wedge (z \vee 1) \wedge (1 \vee \neg z)$

Finally

We have the following reduced number of equations

$$(y \vee z), (1), (\neg y), (y \vee \neg z), (1), (1) \Leftrightarrow \boldsymbol{(y \vee z), (\neg y), (y \vee \neg z)}$$

What if $w = 0, x = 1$

1. Before
   1. $(w \vee x \vee y \vee z) \wedge (w \vee \neg x) \wedge (x \vee \neg y) \wedge (y \vee \neg z) \wedge (z \vee \neg w) \wedge (\neg w \vee \neg z)$
2. After
   1. $(1) \wedge (0) \wedge (1) \wedge (y \vee \neg z) \wedge (1) \wedge (1)$

Finally

We have the following reduced number of equations

$(y \vee z), (1), (\neg y), (y \vee \neg z), (1), (1) \Leftrightarrow \boldsymbol{(y \vee z), (\neg y), (y \vee \neg z)}$

What if $w = 0, x = 1$

1. Before
   1. $(w \vee x \vee y \vee z) \wedge (w \vee \neg x) \wedge (x \vee \neg y) \wedge (y \vee \neg z) \wedge (z \vee \neg w) \wedge (\neg w \vee \neg z)$
2. After
   1. $(1) \wedge (0) \wedge (1) \wedge (y \vee \neg z) \wedge (1) \wedge (1)$

Thus

We have something no satisfiable

$(1) \wedge (0) \wedge (1) \wedge (y \vee \neg z) \wedge (1) \wedge (1) \Leftrightarrow (), (y \vee \neg z)$

Clearly

We prune that part of the search tree.

Note: we use "()≡(0)" to point out to a "empty clause" ruling out satisfiability.

Thus

We have something no satisfiable

$(1) \wedge (0) \wedge (1) \wedge (y \vee \neg z) \wedge (1) \wedge (1) \Leftrightarrow (), (y \vee \neg z)$

Clearly

We prune that part of the search tree.

Note we use "$()\equiv(0)$" to point out to a "empty clause" ruling out satisfiability.

The decisions we need to make in backtracking

First

Which subproblem to expand next.

Second

Which branching variable to use.

Remark

The benefit of backtracking lies in its ability to eliminate portions of the search space.

The decisions we need to make in backtracking

First

Which subproblem to expand next.

Second

Which branching variable to use.

Remark

The benefit of backtracking lies in its ability to eliminate portions of the search space.

The decisions we need to make in backtracking

First

Which subproblem to expand next.

Second

Which branching variable to use.

Remark

The benefit of backtracking lies in its ability to eliminate portions of the search space.

Choosing

Something Notable

A classic strategy:

- You choose the subproblem that contains the smallest clause.
- Then, you branch on a variable in that clause.

Choosing

Something Notable

A classic strategy:

- You choose the subproblem that contains the smallest clause.
- Then, you branch on a variable in that clause.

Then

If the clause is a singleton then at least one of the resulting branches will be terminated.

Choosing

Something Notable

A classic strategy:

- You choose the subproblem that contains the smallest clause.
- Then, you branch on a variable in that clause.

Then

If the clause is a singleton then at least one of the resulting branches will be terminated.

Choosing

Something Notable

A classic strategy:

- You choose the subproblem that contains the smallest clause.
- Then, you branch on a variable in that clause.

Then

If the clause is a singleton then at least one of the resulting branches will be terminated.

The Backtracking Test

The test needs to look at the subproblem to declare quickly if

1. **Failure**: the subproblem has no solution.
2. **Success**: a solution to the subproblem is found.
3. **Uncertainty**.

The Backtracking Test

The test needs to look at the subproblem to declare quickly if

1. **Failure**: the subproblem has no solution.
2. **Success**: a solution to the subproblem is found.
3. **Uncertainty**.

What about SAT

- The test declares failure if there is an empty clause
- The test declares success if there are no clauses
- Uncertainty Otherwise.

The Backtracking Test

The test needs to look at the subproblem to declare quickly if

1. **Failure**: the subproblem has no solution.
2. **Success**: a solution to the subproblem is found.
3. **Uncertainty**.

What about SAT

- The test declares failure if there is an empty clause
- The test declares success if there are no clauses
- Uncertainty Otherwise.

The Backtracking Test

The test needs to look at the subproblem to declare quickly if

1. **Failure**: the subproblem has no solution.
2. **Success**: a solution to the subproblem is found.
3. **Uncertainty**.

What about SAT

- The test declares failure if there is an empty clause
- The test declares success if there are no clauses
- Uncertainty Otherwise.

The Backtracking Test

The test needs to look at the subproblem to declare quickly if

1. **Failure**: the subproblem has no solution.
2. **Success**: a solution to the subproblem is found.
3. **Uncertainty**.

What about SAT

- The test declares failure if there is an empty clause
- The test declares success if there are no clauses
- Uncertainty Otherwise.

The Backtracking Test

The test needs to look at the subproblem to declare quickly if

1. **Failure**: the subproblem has no solution.
2. **Success**: a solution to the subproblem is found.
3. **Uncertainty**.

What about SAT

- The test declares failure if there is an empty clause
- The test declares success if there are no clauses
- Uncertainty Otherwise.

Example

We have the following