An Introduction to Boolean Satisfiability

Mohamed Siala
siala.github.io

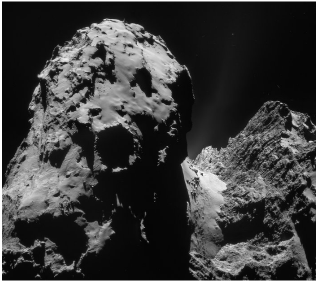
INSA-Toulouse & LAAS-CNRS

January 26, 2023

Context & Introduction

Context: Solving (Very) Hard Combinatorial Problems

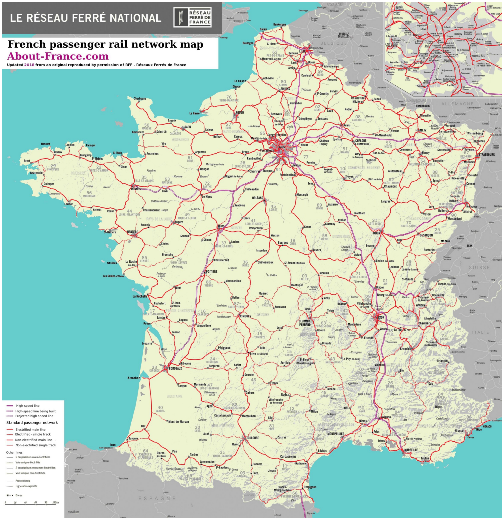
Context: Solving (Very) Hard Combinatorial Problems



https://homepages.laas.fr/ehebrard/rosetta.html

Context: Solving (Very) Hard Combinatorial Problems

Context: Solving (Very) Hard Combinatorial Problems



Context: Solving (Very) Hard Combinatorial Problems

Context: Solving (Very) Hard Combinatorial Problems

Context: Solving (Very) Hard Combinatorial Problems

Context: Solving (Very) Hard Combinatorial Problems

Why this Lecture?

- We are missing job opportunities in optimisation!

Why this Lecture?

- We are missing job opportunities in optimisation!
- SAT as a tool to tackle combinatorial problems

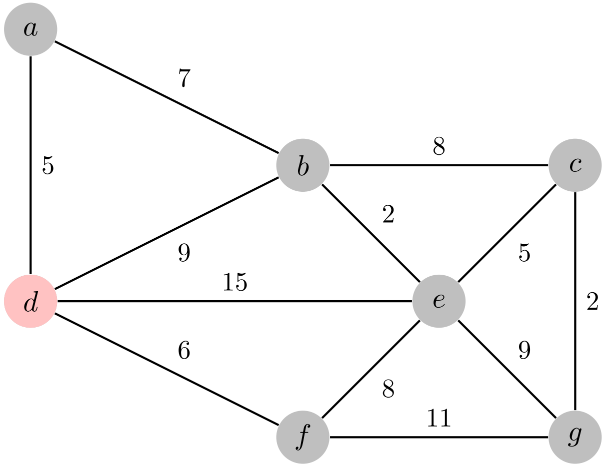
Why this Lecture?

- We are missing job opportunities in optimisation!
- SAT as a tool to tackle combinatorial problems
- We focus in this course on the modelling aspect

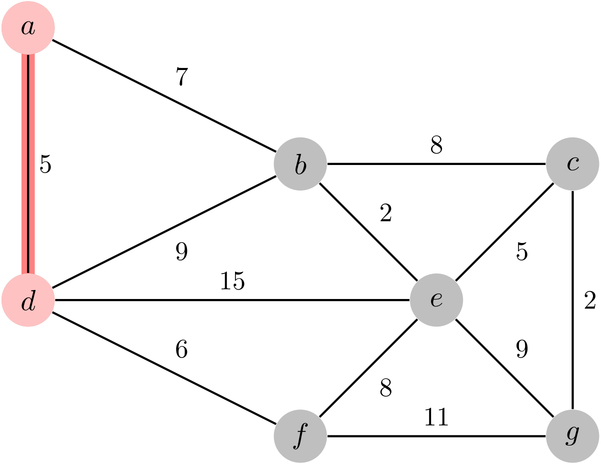
Why this Lecture?

- We are missing job opportunities in optimisation!
- SAT as a tool to tackle combinatorial problems
- We focus in this course on the modelling aspect
- Resources for combinatorial optimisation: Many! a good start would be the online course on discrete optimisation `https://www.coursera.org/learn/discrete-optimization`
- Handbook of Satisfiability - Second Edition - IOS Press, 2021

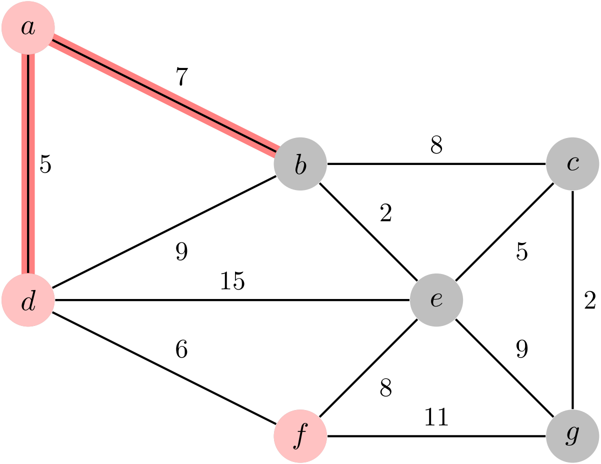
Exemple



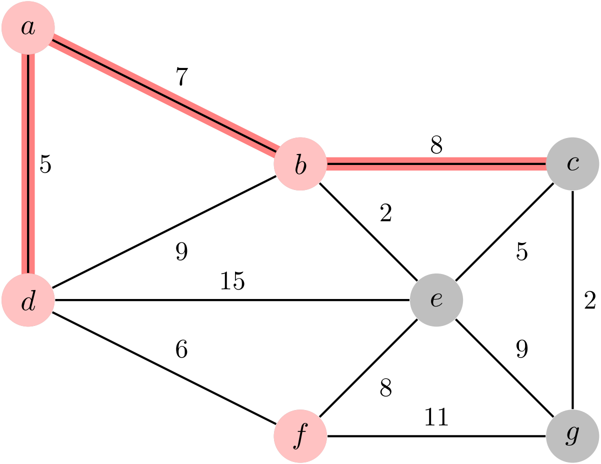
Exemple



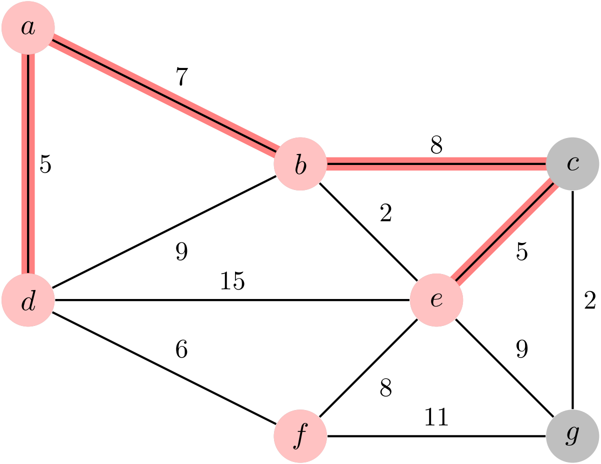
Exemple



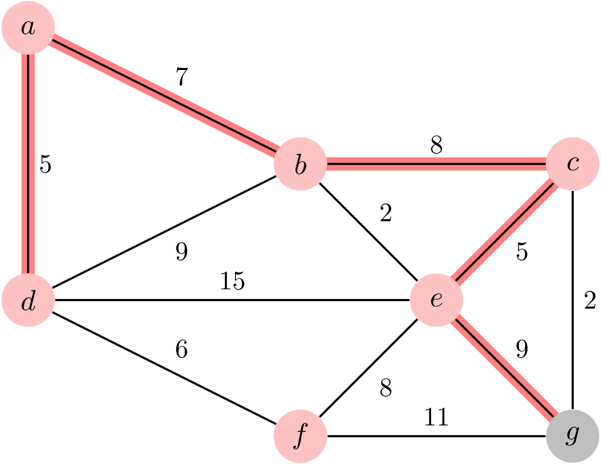
Exemple



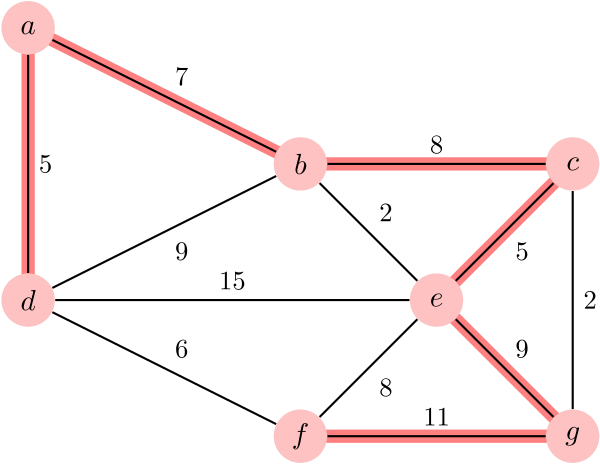
Exemple



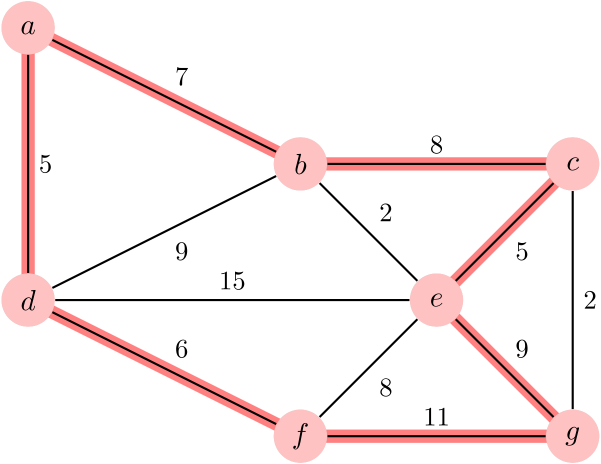
Exemple



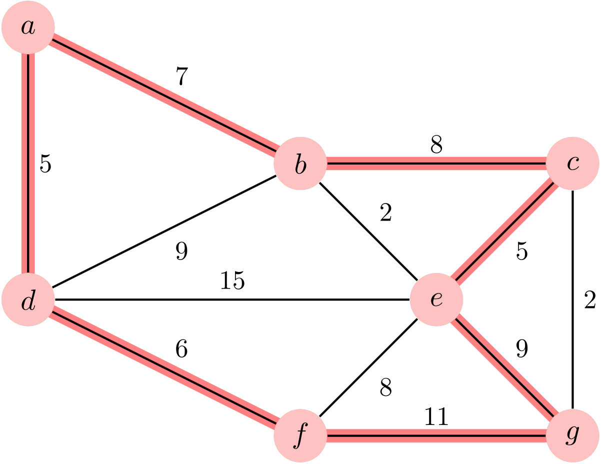
Exemple



Exemple

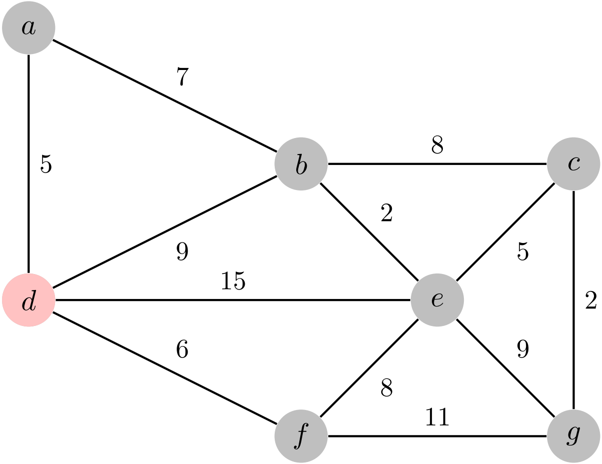


Exemple

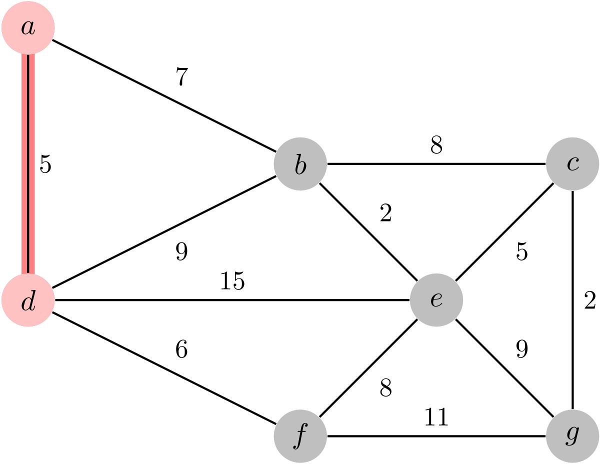


$$--> Cost : 5+7+8+5+9+11+6 = 53Km$$

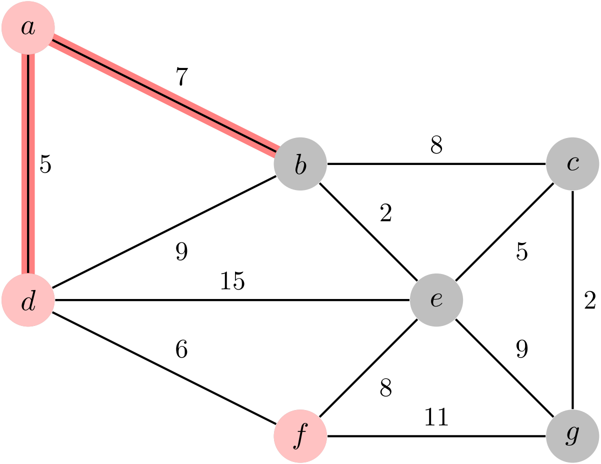
Example



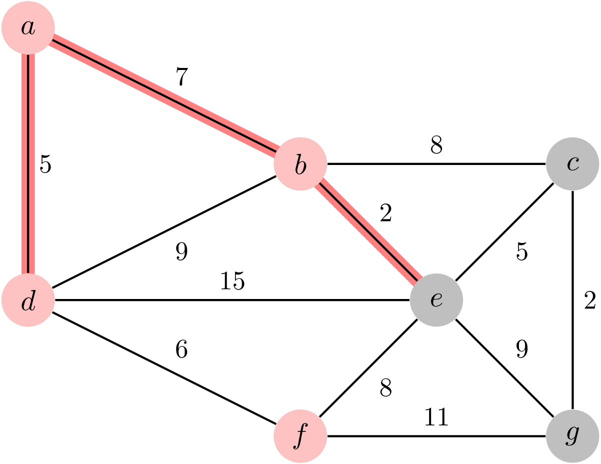
Example



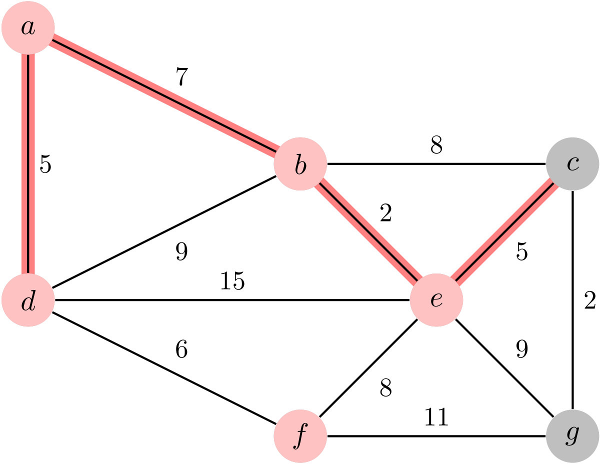
Example



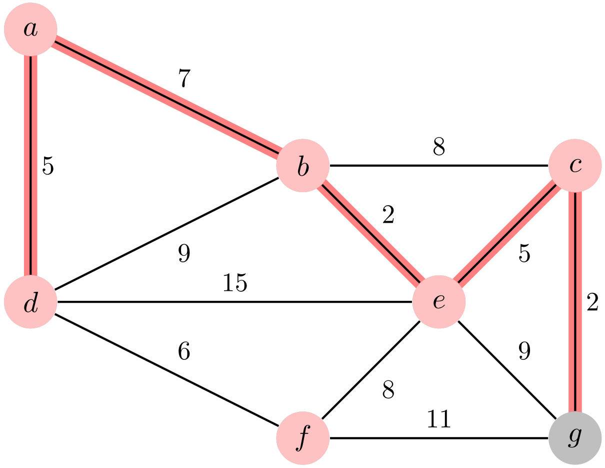
Example



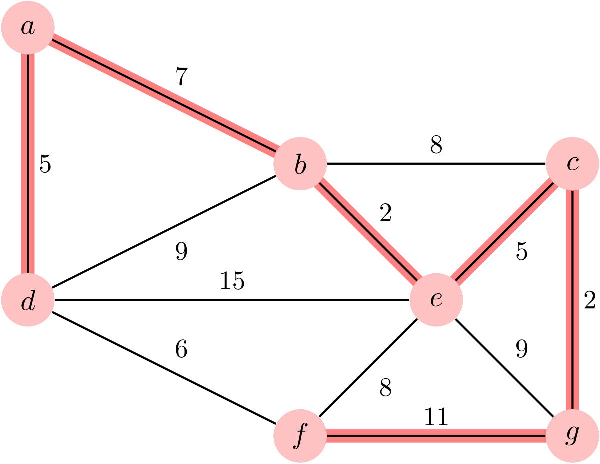
Example



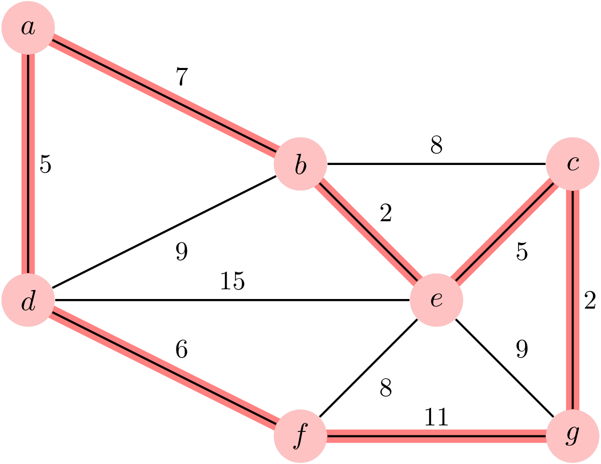
Example



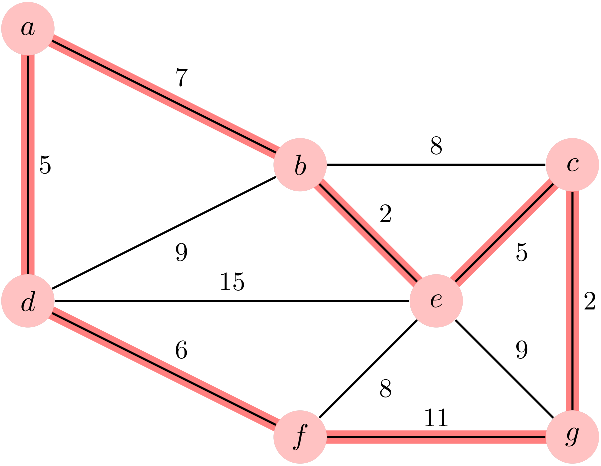
Example



Example



Example



$$--> Cost : 5 + 7 + 2 + 5 + 2 + 11 + 6 = 38Km$$

What if we check all possibilities?

What if we check all possibilities?

- 2 Cities $\rightarrow$1

What if we check all possibilities?

- 2 Cities $\rightarrow$1
- 5 Cities $\rightarrow$24

What if we check all possibilities?

- 2 Cities $\rightarrow$1
- 5 Cities $\rightarrow$24
- 8 Cities $\rightarrow$4032

What if we check all possibilities?

- 2 Cities $\rightarrow$1
- 5 Cities $\rightarrow$24
- 8 Cities $\rightarrow$4032
- 40 Cities

What if we check all possibilities?

- 2 Cities $\rightarrow$1
- 5 Cities $\rightarrow$24
- 8 Cities $\rightarrow$4032
- 40 Cities $\rightarrow 2.10^{46}$ (with a modern machine: 3.10^{27} years!)

What if we check all possibilities?

- 2 Cities $\rightarrow$1
- 5 Cities $\rightarrow$24
- 8 Cities $\rightarrow$4032
- 40 Cities $\rightarrow 2.10^{46}$ (with a modern machine: 3.10^{27} years!)
- 95 Cities, if we use a Planck (the shortest possible time interval that can be measured) processor and fill the universe with one processor per mm^3, we need $3\times$ the age of the universe

What if we check all possibilities?

- 2 Cities $\rightarrow$1
- 5 Cities $\rightarrow$24
- 8 Cities $\rightarrow$4032
- 40 Cities $\rightarrow 2.10^{46}$ (with a modern machine: 3.10^{27} years!)
- 95 Cities, if we use a Planck (the shortest possible time interval that can be measured) processor and fill the universe with one processor per mm^3, we need $3\times$ the age of the universe

The problem is inherently hard. However, the Concorde algorithm can solve instances up to 86 000 cities!

A step back: Problems, Instances, and Algorithms

A step back: Problems, Instances, and Algorithms

- A problem is a question that associates an input to an output

A step back: Problems, Instances, and Algorithms

- A problem is a question that associates an input to an output
- Many instances (instantiation of the input) for the same problem

A step back: Problems, Instances, and Algorithms

- A problem is a question that associates an input to an output
- Many instances (instantiation of the input) for the same problem
- Many algorithms (methodologies) to solve the same problem

A step back: Problems, Instances, and Algorithms

- A problem is a question that associates an input to an output
- Many instances (instantiation of the input) for the same problem
- Many algorithms (methodologies) to solve the same problem

Solving Methodologies

1. Adhoc methods
   1. Specific exact algorithm
   2. Heuristic method
   3. Meta-heuristic (genetic algorithms, ant colony, ..)
2. Declarative Approaches
   1. (Mixed) Integer Programming,
   2. Constraint Programming
   3. Boolean Satisfiability (SAT)
   4. ...

Why Declarative Approaches?

- They are problem independent! The user models the problem in a specific language and the solver does the job!
- Very active community

Solving Methodologies

Solving Methodologies

1. Adhoc methods

Solving Methodologies

1. Adhoc methods

Solving Methodologies

1. Adhoc methods
   1. Specific exact algorithm

Solving Methodologies

1. Adhoc methods
    1. Specific exact algorithm
    2. Heuristic method

Solving Methodologies

1. Adhoc methods
   1. Specific exact algorithm
   2. Heuristic method
   3. Meta-heuristic (genetic algorithms, ant colony, ..)
2. Declarative Approaches

Solving Methodologies

1. Adhoc methods
    1. Specific exact algorithm
    2. Heuristic method
    3. Meta-heuristic (genetic algorithms, ant colony, ..)
2. Declarative Approaches
    1. (Mixed) Integer Programming,

Solving Methodologies

1. Adhoc methods
   1. Specific exact algorithm
   2. Heuristic method
   3. Meta-heuristic (genetic algorithms, ant colony, ..)
2. Declarative Approaches
   1. (Mixed) Integer Programming,
   2. Constraint Programming

Solving Methodologies

1. Adhoc methods
   1. Specific exact algorithm
   2. Heuristic method
   3. Meta-heuristic (genetic algorithms, ant colony, ..)
2. Declarative Approaches
   1. (Mixed) Integer Programming,
   2. Constraint Programming
   3. Boolean Satisfiability (SAT)
   4. ...

Why Declarative Approaches?

Solving Methodologies

1. Adhoc methods
   1. Specific exact algorithm
   2. Heuristic method
   3. Meta-heuristic (genetic algorithms, ant colony, ..)
2. Declarative Approaches
   1. (Mixed) Integer Programming,
   2. Constraint Programming
   3. Boolean Satisfiability (SAT)
   4. ...

Why Declarative Approaches?

- They are problem independent! The user models the problem in a specific language and the solver does the job!

Solving Methodologies

1. Adhoc methods
   1. Specific exact algorithm
   2. Heuristic method
   3. Meta-heuristic (genetic algorithms, ant colony, ..)
2. Declarative Approaches
   1. (Mixed) Integer Programming,
   2. Constraint Programming
   3. Boolean Satisfiability (SAT)
   4. ...

Why Declarative Approaches?

- They are problem independent! The user models the problem in a specific language and the solver does the job!
- Very active community

Complexity: How to Evaluate Algorithms

Complexity: How to Evaluate Algorithms

- Complexity: a measure to analyze/classify algorithms based on the amount of resources required (Time and Memory)

Complexity: How to Evaluate Algorithms

- Complexity: a measure to analyze/classify algorithms based on the amount of resources required (Time and Memory)
- Time Complexity: number of operations as a function of the size of the input

Complexity: How to Evaluate Algorithms

- Complexity: a measure to analyze/classify algorithms based on the amount of resources required (Time and Memory)
- Time Complexity: number of operations as a function of the size of the input
- Space Complexity: memory occupied by the algorithm as a function of the size of the input

Complexity: How to Evaluate Algorithms

- Complexity: a measure to analyze/classify algorithms based on the amount of resources required (Time and Memory)
- Time Complexity: number of operations as a function of the size of the input
- Space Complexity: memory occupied by the algorithm as a function of the size of the input
- The evaluation is made usually by reasoning about the worst case.

Complexity: How to Evaluate Algorithms

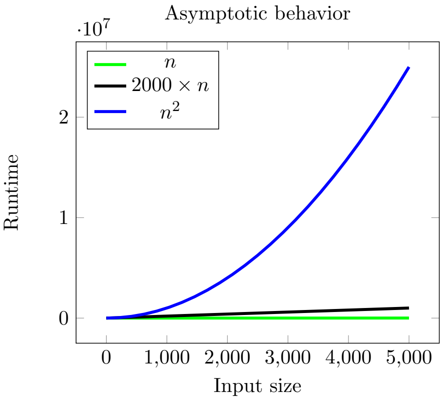
- Complexity: a measure to analyze/classify algorithms based on the amount of resources required (Time and Memory)
- Time Complexity: number of operations as a function of the size of the input
- Space Complexity: memory occupied by the algorithm as a function of the size of the input
- The evaluation is made usually by reasoning about the worst case.
- The analysis is given with regard to the asymptotic behaviour

Complexity: How to Evaluate Algorithms

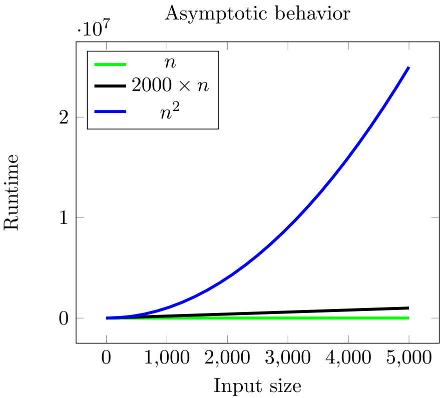
- Complexity: a measure to analyze/classify algorithms based on the amount of resources required (Time and Memory)
- Time Complexity: number of operations as a function of the size of the input
- Space Complexity: memory occupied by the algorithm as a function of the size of the input
- The evaluation is made usually by reasoning about the worst case.
- The analysis is given with regard to the asymptotic behaviour

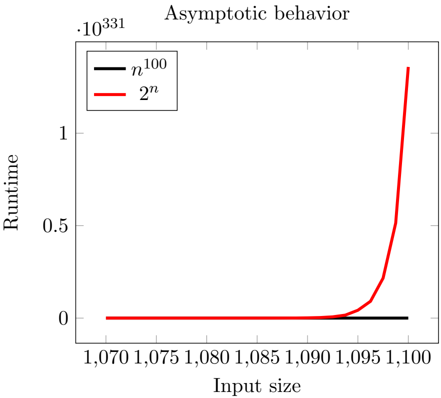
Asymptotic behaviour

Asymptotic behaviour



Asymptotic behaviour





- If f is polynomial and g is exponential then $f \in O(g)$.
  For instance $n^{10000} \in O(2^n)$

- If f is polynomial and g is exponential then $f \in O(g)$.
  For instance $n^{10000} \in O(2^n)$
- Convention:
  - Easy/Tractable Problem: We know a polynomial time algorithm to solve the problem
  - Hard/Intractable: No known polynomial algorithm

- If f is polynomial and g is exponential then $f \in O(g)$. For instance $n^{10000} \in O(2^n)$
- Convention:
  - Easy/Tractable Problem: We know a polynomial time algorithm to solve the problem
  - Hard/Intractable: No known polynomial algorithm
- Example: The sorting problem is easy because we have an algorithm that runs in the worst case in $O(n \log(n))$

- If f is polynomial and g is exponential then $f \in O(g)$.
  For instance $n^{10000} \in O(2^n)$
- Convention:
  - Easy/Tractable Problem: We know a polynomial time algorithm to solve the problem
  - Hard/Intractable: No known polynomial algorithm
- Example: The sorting problem is easy because we have an algorithm that runs in the worst case in $O(n \log(n))$
- What if we don't know if a problem has a polynomial time algorithm?

Classes of problems

Classes of problems

- **P** is the class of problems that are **solvable** in polynomial time (easy problems)

Classes of problems

- **P** is the class of problems that are **solvable** in polynomial time (easy problems)
- **NP** is the class of problems that are **verifiable** in polynomial time algorithm: Give me a candidate solution $\mathcal{S}$ and one can tell in polynomial time if $\mathcal{S}$ is a solution

Classes of problems

- **P** is the class of problems that are **solvable** in polynomial time (easy problems)
- **NP** is the class of problems that are **verifiable** in polynomial time algorithm: Give me a candidate solution $\mathcal{S}$ and one can tell in polynomial time if $\mathcal{S}$ is a solution
- We know that $P \in NP$ (if you can solve in n^d then you can verify in n^d)

Classes of problems

- **P** is the class of problems that are **solvable** in polynomial time (easy problems)
- **NP** is the class of problems that are **verifiable** in polynomial time algorithm: Give me a candidate solution $\mathcal{S}$ and one can tell in polynomial time if $\mathcal{S}$ is a solution
- We know that $P \in NP$ (if you can solve in n^d then you can verify in n^d)
- For many Problems in NP, we don't know if a polynomial time algorithm exists. Is P=NP?

The Boolean Satisfiability Problem (SAT)

The Boolean Satisfiability Problem (SAT)

Definitions

- Atoms (Boolean variables): x_1, x_2, ...

The Boolean Satisfiability Problem (SAT)

Definitions

- Atoms (Boolean variables): x_1, x_2, ...
- Literal: $x_1, \neg x_1$

The Boolean Satisfiability Problem (SAT)

Definitions

- Atoms (Boolean variables): x_1, x_2, ...
- Literal: $x_1, \neg x_1$
- Clauses: a clause is a disjunction of literals

The Boolean Satisfiability Problem (SAT)

Definitions

- Atoms (Boolean variables): x_1, x_2, ...
- Literal: $x_1, \neg x_1$
- Clauses: a clause is a disjunction of literals
- Example of clause: $(\neg x_1 \vee \neg x_4 \vee x_7)$

The Boolean Satisfiability Problem (SAT)

Definitions

- Atoms (Boolean variables): x_1, x_2, ...
- Literal: $x_1, \neg x_1$
- Clauses: a clause is a disjunction of literals
- Example of clause: $(\neg x_1 \vee \neg x_4 \vee x_7)$
- Propositional formula Φ given in a `Conjunctive Normal Form` (`CNF`) $\Phi : c_1 \wedge .. \wedge c_n$

The Boolean Satisfiability Problem (SAT)

Definitions

- Atoms (Boolean variables): x_1, x_2, ...
- Literal: $x_1, \neg x_1$
- Clauses: a clause is a disjunction of literals
- Example of clause: $(\neg x_1 \vee \neg x_4 \vee x_7)$
- Propositional formula Φ given in a `Conjunctive Normal Form` (`CNF`) $\Phi : c_1 \wedge .. \wedge c_n$

The Boolean Satisfiability Problem (SAT)

Definitions

- Atoms (Boolean variables): x_1, x_2, ...
- Literal: $x_1, \neg x_1$
- Clauses: a clause is a disjunction of literals
- Example of clause: $(\neg x_1 \vee \neg x_4 \vee x_7)$
- Propositional formula Φ given in a `Conjunctive Normal Form` (`CNF`) $\Phi : c_1 \wedge .. \wedge c_n$

Given a set of Boolean variables $x_1, \ldots x_n$ and a CNF formula Φ over $x_1, \ldots x_n$, the Boolean Satisfiability problem (SAT) is to find an assignment of the variables that satisfies all the clauses.

Example

Example

$x \vee \neg y \vee z$

$\neg x \vee \neg z$

$y \vee w$

$\neg w \vee \neg x$

Example

$x \vee \neg y \vee z$

$\neg x \vee \neg z$

$y \vee w$

$\neg w \vee \neg x$

A possible solution:

$$x \leftarrow 1; y \leftarrow 1; z \leftarrow 0; w \leftarrow 0$$

Why SAT?

Why SAT?

- SAT is the first problem that is shown to be in the class NP-Complete (the class of the 'hardest' problems in NP):
  - Any problem in NP can be reduced polynomially to SAT
  - If you can solve SAT in polynomial time, call me straight away!
  - If you find a polynomial time algorithm to solve SAT you solve the $P = NP$? question

Why SAT?

- SAT is the first problem that is shown to be in the class NP-Complete (the class of the 'hardest' problems in NP):
  - Any problem in NP can be reduced polynomially to SAT
  - If you can solve SAT in polynomial time, call me straight away!
  - If you find a polynomial time algorithm to solve SAT you solve the $P = NP$? question
- It is considered today as a powerful technology to solve computational problems

Why SAT?

- SAT is the first problem that is shown to be in the class NP-Complete (the class of the 'hardest' problems in NP):
  - Any problem in NP can be reduced polynomially to SAT
  - If you can solve SAT in polynomial time, call me straight away!
  - If you find a polynomial time algorithm to solve SAT you solve the $P = NP$? question
- It is considered today as a powerful technology to solve computational problems
- Huge practical improvements in the past 2 decades or so

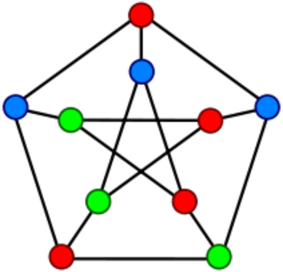
Examples of Applications

- AI Planning
- Scheduling
- Software verification
- Machine learning
  - Robustness
  - Synthesis
  - Verification
- Mathematical Proofs! `https://news.cnrs.fr/articles/the-longest-proof-in-the-history-of-mathematics`
- Timetabling
- ...

Modelling in SAT

The example of Graph Colouring

- Graph Coloring is a well known combinatorial problem that has many applications (in particular in scheduling problems)
- Let $G = (V, E)$ be an undirected graph where V is a set of n vertices and E is a set of m edges. Is it possible to colour the graph with k colours such that no two adjacent nodes share the same colour?



Modelling in SAT: The Example of Graph Coloring

Modelling in SAT: The Example of Graph Coloring

- Propose a SAT model for this problem
- There is no need to explicitly encode $x \rightarrow y$ since it is equivalent $\neg x \vee y$

Modelling in SAT: The Example of Graph Coloring

- Propose a SAT model for this problem
- There is no need to explicitly encode $x \rightarrow y$ since it is equivalent $\neg x \vee y$

The Example of Graph Coloring: A Possible Model

The Example of Graph Coloring: A Possible Model

Let x_i^k be the Boolean variable that is True iff node i is coloured with the colour k.

The Example of Graph Coloring: A Possible Model

Let x_i^k be the Boolean variable that is True iff node i is coloured with the colour k.

- Each node has to be colored with at least one color:

$$\forall i \in [1, n], x_i^1 \vee x_i^2 \dots x_i^k$$

The Example of Graph Coloring: A Possible Model

Let x_i^k be the Boolean variable that is True iff node i is coloured with the colour k.

- Each node has to be colored with at least one color:

$$\forall i \in [1, n], x_i^1 \lor x_i^2 \ldots x_i^k$$

- If a node is coloured with a colour a, the other colours are forbidden:

$$\forall i \in [1, n], \forall a \neq b \in [1, k], : \neg x_i^a \lor \neg x_i^b$$

The Example of Graph Coloring: A Possible Model

Let x_i^k be the Boolean variable that is True iff node i is coloured with the colour k.

- Each node has to be colored with at least one color:

$$\forall i \in [1, n], x_i^1 \vee x_i^2 \ldots x_i^k$$

- If a node is coloured with a colour a, the other colours are forbidden:

$$\forall i \in [1, n], \forall a \neq b \in [1, k], : \neg x_i^a \vee \neg x_i^b$$

  (This is a translation of $x_i^a \rightarrow \neg x_i^b$)

The Example of Graph Coloring: A Possible Model

Let x_i^k be the Boolean variable that is True iff node i is coloured with the colour k.

- Each node has to be colored with at least one color:

$$\forall i \in [1, n], x_i^1 \vee x_i^2 \ldots x_i^k$$

- If a node is coloured with a colour a, the other colours are forbidden:

$$\forall i \in [1, n], \forall a \neq b \in [1, k], : \neg x_i^a \vee \neg x_i^b$$

  (This is a translation of $x_i^a \rightarrow \neg x_i^b$)
- Forbid two nodes that share an edge to be coloured with the same colour

$$\forall \{i, j\} \in E, \forall a \in [1, k] : \neg x_i^a \vee \neg x_j^a$$

  (This is a translation of $x_i^a \rightarrow \neg x_j^a$)

The Example of Graph Coloring: The Model Size

What is the (space) size of the model?

- $n \times k$ Boolean variables
- Constraints form 1: n clauses with k literals each
- Constraints form 2: $n \times k^2$ binary clauses
- Constraints form 3: $m \times k$ binary clauses

The Example of Graph Coloring: The Minimization Version

- Propose a method that uses SAT for the minimisation version of the problem. That is, given $G = (V, E)$, we seek to find the minimum value of k to satisfy the colouring requirements.

A Straightforward Approach

A Straightforward Approach

- Find a valid upper bound UB and a lower bound LB for k

A Straightforward Approach

- Find a valid upper bound UB and a lower bound LB for k
- Run iteratively the decision version until converging to the optimal value

A Straightforward Approach

- Find a valid upper bound UB and a lower bound LB for k
- Run iteratively the decision version until converging to the optimal value
- Let's call $SAT(V, E, K)$ the SAT model of the decision version of the problem (i.e., can we find a valid colouring of $G(V, E)$ with k colours). Use $SAT(V, E, K)$ as an oracle within an iterative search. For instance:

A Straightforward Approach

- Find a valid upper bound UB and a lower bound LB for k
- Run iteratively the decision version until converging to the optimal value
- Let's call $SAT(V, E, K)$ the SAT model of the decision version of the problem (i.e., can we find a valid colouring of $G(V, E)$ with k colours). Use $SAT(V, E, K)$ as an oracle within an iterative search. For instance:
  - **Decreasing linear Search:** Run iteratively $SAT(V, E, UB - 1), SAT(V, E, UB - 2), \dots$ until the problem is unsatisfiable. The last satisfiable value of k is the optimal value

A Straightforward Approach

- Find a valid upper bound UB and a lower bound LB for k
- Run iteratively the decision version until converging to the optimal value
- Let's call $SAT(V, E, K)$ the SAT model of the decision version of the problem (i.e., can we find a valid colouring of $G(V, E)$ with k colours). Use $SAT(V, E, K)$ as an oracle within an iterative search. For instance:
  - **Decreasing linear Search:** Run iteratively $SAT(V, E, UB-1), SAT(V, E, UB-2), \ldots$ until the problem is unsatisfiable. The last satisfiable value of k is the optimal value
  - **Binary search:** Run iteratively $SAT(V, E, z)$ as long as $UB > LB$ where $z = \lceil (UB - LB)/2 \rceil$. If the result is satisfiable, then and $UB \leftarrow z$ otherwise $LB \leftarrow z$

Upper/Lower Bound?

Upper/Lower Bound?

- Upper bound: For instance, we can run the following iterative greedy algorithm:

Upper/Lower Bound?

- Upper bound: For instance, we can run the following iterative greedy algorithm:
  - Each vertex v is considered non-coloured and has a portfolio S_v of available colours. The set is initially $\{1, 2, \dots n\}$ for each vertex

Upper/Lower Bound?

- Upper bound: For instance, we can run the following iterative greedy algorithm:
  - Each vertex v is considered non-coloured and has a portfolio S_v of available colours. The set is initially $\{1, 2, \ldots n\}$ for each vertex
  - At each iteration, look for a non-coloured vertex v that has the greatest number of non coloured neighbours. Colour it with the smallest colour in S_v and remove its colour from all its neighbours.

Upper/Lower Bound?

- Upper bound: For instance, we can run the following iterative greedy algorithm:
    - Each vertex v is considered non-coloured and has a portfolio S_v of available colours. The set is initially $\{1, 2, \dots n\}$ for each vertex
    - At each iteration, look for a non-coloured vertex v that has the greatest number of non coloured neighbours. Colour it with the smallest colour in S_v and remove its colour from all its neighbours.
    - The resulting colouring is valid and the upper bound is the number of different colours used.

Upper/Lower Bound?

- Upper bound: For instance, we can run the following iterative greedy algorithm:
  - Each vertex v is considered non-coloured and has a portfolio S_v of available colours. The set is initially $\{1, 2, \ldots n\}$ for each vertex
  - At each iteration, look for a non-coloured vertex v that has the greatest number of non coloured neighbours. Colour it with the smallest colour in S_v and remove its colour from all its neighbours.
  - The resulting colouring is valid and the upper bound is the number of different colours used.
  - The run time complexity is $O(n^2 \times m)$

Upper/Lower Bound?

- Upper bound: For instance, we can run the following iterative greedy algorithm:
  - Each vertex v is considered non-coloured and has a portfolio S_v of available colours. The set is initially $\{1, 2, \ldots n\}$ for each vertex
  - At each iteration, look for a non-coloured vertex v that has the greatest number of non coloured neighbours. Colour it with the smallest colour in S_v and remove its colour from all its neighbours.
  - The resulting colouring is valid and the upper bound is the number of different colours used.
  - The run time complexity is $O(n^2 \times m)$
- Lower bound: one can simply consider 2 as long as there is an edge. A more advanced one is to look for a clique in the graph.

Upper/Lower Bound?

- Upper bound: For instance, we can run the following iterative greedy algorithm:
  - Each vertex v is considered non-coloured and has a portfolio S_v of available colours. The set is initially $\{1, 2, \dots n\}$ for each vertex
  - At each iteration, look for a non-coloured vertex v that has the greatest number of non coloured neighbours. Colour it with the smallest colour in S_v and remove its colour from all its neighbours.
  - The resulting colouring is valid and the upper bound is the number of different colours used.
  - The run time complexity is $O(n^2 \times m)$
- Lower bound: one can simply consider 2 as long as there is an edge. A more advanced one is to look for a clique in the graph.

Upper/Lower Bound?

- Upper bound: For instance, we can run the following iterative greedy algorithm:
  - Each vertex v is considered non-coloured and has a portfolio S_v of available colours. The set is initially $\{1, 2, \ldots n\}$ for each vertex
  - At each iteration, look for a non-coloured vertex v that has the greatest number of non coloured neighbours. Colour it with the smallest colour in S_v and remove its colour from all its neighbours.
  - The resulting colouring is valid and the upper bound is the number of different colours used.
  - The run time complexity is $O(n^2 \times m)$
- Lower bound: one can simply consider 2 as long as there is an edge. A more advanced one is to look for a clique in the graph.
- An alternative approach is to look for valid theoretical bounds in the literature.

Modelling Cardinality Constraints

- A cardinality constraint takes as input a sequence of Boolean variables $[x_1, \ldots, x_n]$ and an integer k and enforces

$$\sum_{1}^{n} x_i \leq k$$

- Cardinality constraints are everywhere!
- There exist many ways in the literature to encode such constraints. See for instance `https://www.carstensinz.de/papers/CP-2005.pdf`

Quadratic encoding for $\sum_1^n x_i = 1$

- At least one constraint:

$$x_1 \vee x_2 \ldots \vee x_n$$

- at most one constraint:

$$\forall i, j : \neg x_i \vee \neg x_j$$

This generates one clause of size n and (n^2) binary clauses without introducing additional variables.

Linear encoding for $\sum_1^n x_i = 1$

A sequence of Boolean variables $[y_1, \ldots, y_n]$ is introduced such that $\forall i \in [1, n], y_i$ is true iff $\sum_{l=1}^{l=i} x_l = 1$. The set of clauses for the encoding is the following:

$$x_1 \vee x_2 \ldots \vee x_n$$

$$y_n^1$$

$$\forall i \in [1, n-1] : y_i \rightarrow y_{i+1}$$

$$\forall i \in [1, n-1] : y_i \rightarrow \neg x_{i+1}$$

$$\forall i \in [1, n] : x_i \rightarrow y_i$$

Size: n new variables, 1 $n-$ary clause and $3 \times n$ binary clauses,

Encoding for $\sum_1^n x_i \geq k$

Encoding for $\sum_1^n x_i \geq k$

- New variables: $\forall z \in [0, k], \forall i \in [1, n], y_i^z \iff \sum_{l=1}^{l=i} x_l \geq z$

Encoding for $\sum_1^n x_i \geq k$

- New variables: $\forall z \in [0, k], \forall i \in [1, n], y_i^z \iff \sum_{l=1}^{l=i} x_l \geq z$
- $y_1^0 \leftarrow 1$

Encoding for $\sum_1^n x_i \geq k$

- New variables: $\forall z \in [0, k], \forall i \in [1, n], y_i^z \iff \sum_{l=1}^{l=i} x_l \geq z$
- $y_1^0 \leftarrow 1$
- $y_1^1 \leftarrow x_1$

Encoding for $\sum_1^n x_i \geq k$

- New variables: $\forall z \in [0, k], \forall i \in [1, n], y_i^z \iff \sum_{l=1}^{l=i} x_l \geq z$
- $y_1^0 \leftarrow 1$
- $y_1^1 \leftarrow x_1$
- $y_1^2 \leftarrow 0$

Encoding for $\sum_1^n x_i \geq k$

- New variables: $\forall z \in [0, k], \forall i \in [1, n], y_i^z \iff \sum_{l=1}^{l=i} x_l \geq z$
- $y_1^0 \leftarrow 1$
- $y_1^1 \leftarrow x_1$
- $y_1^2 \leftarrow 0$
- $y_n^k \leftarrow 1$

Encoding for $\sum_1^n x_i \geq k$

- New variables: $\forall z \in [0, k], \forall i \in [1, n], y_i^z \iff \sum_{l=1}^{l=i} x_l \geq z$
- $y_1^0 \leftarrow 1$
- $y_1^1 \leftarrow x_1$
- $y_1^2 \leftarrow 0$
- $y_n^k \leftarrow 1$
- Vertical relationship: $\forall i \in [1, n], \forall z \in [1, k-1] : y_i^{z+1} \rightarrow y_i^z$

Encoding for $\sum_1^n x_i \geq k$

- New variables: $\forall z \in [0,k], \forall i \in [1,n], y_i^z \iff \sum_{l=1}^{l=i} x_l \geq z$
- $y_1^0 \leftarrow 1$
- $y_1^1 \leftarrow x_1$
- $y_1^2 \leftarrow 0$
- $y_n^k \leftarrow 1$
- Vertical relationship: $\forall i \in [1,n], \forall z \in [1,k-1] : y_i^{z+1} \rightarrow y_i^z$
- Horizontal relationship: $\forall i \in [1,n-1], \forall z \in [0,k] : y_i^z \rightarrow y_{i+1}^z$

Encoding for $\sum_1^n x_i \geq k$

- New variables: $\forall z \in [0, k], \forall i \in [1, n], y_i^z \iff \sum_{l=1}^{l=i} x_l \geq z$
- $y_1^0 \leftarrow 1$
- $y_1^1 \leftarrow x_1$
- $y_1^2 \leftarrow 0$
- $y_n^k \leftarrow 1$
- Vertical relationship: $\forall i \in [1, n], \forall z \in [1, k-1] : y_i^{z+1} \rightarrow y_i^z$
- Horizontal relationship: $\forall i \in [1, n-1], \forall z \in [0, k] : y_i^z \rightarrow y_{i+1}^z$
- Bound the shape: $\neg y_{i-1}^z \rightarrow \neg y_i^{z+1}$

Encoding for $\sum_1^n x_i \geq k$

- New variables: $\forall z \in [0,k], \forall i \in [1,n], y_i^z \iff \sum_{l=1}^{l=i} x_l \geq z$
- $y_1^0 \leftarrow 1$
- $y_1^1 \leftarrow x_1$
- $y_1^2 \leftarrow 0$
- $y_n^k \leftarrow 1$
- Vertical relationship: $\forall i \in [1,n], \forall z \in [1,k-1] : y_i^{z+1} \rightarrow y_i^z$
- Horizontal relationship: $\forall i \in [1,n-1], \forall z \in [0,k] : y_i^z \rightarrow y_{i+1}^z$
- Bound the shape: $\neg y_{i-1}^z \rightarrow \neg y_i^{z+1}$
- Increment the count: $y_{i-1}^z \wedge x_i \rightarrow y_i^{z+1}$

Encoding for $\sum_1^n x_i \geq k$

- New variables: $\forall z \in [0,k], \forall i \in [1,n], y_i^z \iff \sum_{l=1}^{l=i} x_l \geq z$
- $y_1^0 \leftarrow 1$
- $y_1^1 \leftarrow x_1$
- $y_1^2 \leftarrow 0$
- $y_n^k \leftarrow 1$
- Vertical relationship: $\forall i \in [1,n], \forall z \in [1,k-1] : y_i^{z+1} \rightarrow y_i^z$
- Horizontal relationship: $\forall i \in [1,n-1], \forall z \in [0,k] : y_i^z \rightarrow y_{i+1}^z$
- Bound the shape: $\neg y_{i-1}^z \rightarrow \neg y_i^{z+1}$
- Increment the count: $y_{i-1}^z \wedge x_i \rightarrow y_i^{z+1}$
- Do not Increment: $\neg y_{i-1}^z \wedge \neg x_i \rightarrow \neg y_i^z$

Encoding for $\sum_1^n x_i \geq k$

Size of the encoding:

- $\Theta(n \times k)$ variables
- $\Theta(n + k)$ unary clauses
- $\Theta(n \times k)$ binary clauses
- $\Theta(n \times k)$ ternary clauses

Encoding for $\sum_1^n x_i = k$?

Encoding for $\sum_1^n x_i = k$?

Encoding for $\sum_1^n x_i = k$?

- Encode $\sum_1^n x_i \geq k+1$

Encoding for $\sum_1^n x_i = k$?

- Encode $\sum_1^n x_i \geq k+1$
- Add y_n^k
- Replace y_n^{k+1} by $\neg y_n^{k+1}$
- The size of the encoding is the same as $\sum_1^n x_i \geq k$ (asymptotically)

Linear encoding for $\sum_1^n x_i \leq k$?

Linear encoding for $\sum_1^n x_i \leq k$?

- Encode $\sum_1^n x_i \geq k+1$
- Replace y_n^{k+1} by $\neg y_n^{k+1}$
- The size of the encoding is the same as $\sum_1^n x_i \geq k$ (asymptotically)

Linear encoding for $a \leq \sum_1^n x_i \leq b$?

Linear encoding for $a \leq \sum_1^n x_i \leq b$?

- Encode $\sum_1^n x_i \leq b$

Linear encoding for $a \leq \sum_1^n x_i \leq b$?

- Encode $\sum_1^n x_i \leq b$
- $\sum_1^n x_i \geq a$ with the same additional variables
- The size of the encoding is the same as $\sum_1^n x_i \geq k$ (asymptotically)

Extensions: MaxSAT

Extensions: MaxSAT

- MaxSAT is an optimisation extension of SAT where some clauses are "hard" (must be satisfied) and others are "soft" (can be violated).

Extensions: MaxSAT

- MaxSAT is an optimisation extension of SAT where some clauses are "hard" (must be satisfied) and others are "soft" (can be violated).
- The task is to find an assignment of the variables that satisfies the hard clauses and maximises the number of "soft" clauses

Extensions: MaxSAT

- MaxSAT is an optimisation extension of SAT where some clauses are "hard" (must be satisfied) and others are "soft" (can be violated).
- The task is to find an assignment of the variables that satisfies the hard clauses and maximises the number of "soft" clauses
- MaxSAT:

Extensions: MaxSAT

- MaxSAT is an optimisation extension of SAT where some clauses are "hard" (must be satisfied) and others are "soft" (can be violated).
- The task is to find an assignment of the variables that satisfies the hard clauses and maximises the number of "soft" clauses
- MaxSAT:
  - Variables: Booleans, Clauses: hard and soft clauses

Extensions: MaxSAT

- MaxSAT is an optimisation extension of SAT where some clauses are "hard" (must be satisfied) and others are "soft" (can be violated).
- The task is to find an assignment of the variables that satisfies the hard clauses and maximises the number of "soft" clauses
- MaxSAT:
  - Variables: Booleans, Clauses: hard and soft clauses

Extensions: MaxSAT

- MaxSAT is an optimisation extension of SAT where some clauses are "hard" (must be satisfied) and others are "soft" (can be violated).
- The task is to find an assignment of the variables that satisfies the hard clauses and maximises the number of "soft" clauses
- MaxSAT:
  - Variables: Booleans, Clauses: hard and soft clauses
  - Maximisation problem: Is there an assignment of the variables that satisfy all the hard clauses, and maximises the number of satisfied soft clauses?

Extensions: MaxSAT

- MaxSAT is an optimisation extension of SAT where some clauses are "hard" (must be satisfied) and others are "soft" (can be violated).
- The task is to find an assignment of the variables that satisfies the hard clauses and maximises the number of "soft" clauses
- MaxSAT:
  - Variables: Booleans, Clauses: hard and soft clauses
  - Maximisation problem: Is there an assignment of the variables that satisfy all the hard clauses, and maximises the number of satisfied soft clauses?
- Weighted MaxSAT: Extension of MaxSAT where every soft clause is associated with a weight

Extensions: MaxSAT

- MaxSAT is an optimisation extension of SAT where some clauses are "hard" (must be satisfied) and others are "soft" (can be violated).
- The task is to find an assignment of the variables that satisfies the hard clauses and maximises the number of "soft" clauses
- MaxSAT:
  - Variables: Booleans, Clauses: hard and soft clauses
  - Maximisation problem: Is there an assignment of the variables that satisfy all the hard clauses, and maximises the number of satisfied soft clauses?
- Weighted MaxSAT: Extension of MaxSAT where every soft clause is associated with a weight
- Objective: satisfy hard clauses and maximise the weighted sum of satisfied soft clauses.

Extensions: MaxSAT

- MaxSAT is an optimisation extension of SAT where some clauses are "hard" (must be satisfied) and others are "soft" (can be violated).
- The task is to find an assignment of the variables that satisfies the hard clauses and maximises the number of "soft" clauses
- MaxSAT:
    - Variables: Booleans, Clauses: hard and soft clauses
    - Maximisation problem: Is there an assignment of the variables that satisfy all the hard clauses, and maximises the number of satisfied soft clauses?
- Weighted MaxSAT: Extension of MaxSAT where every soft clause is associated with a weight
- Objective: satisfy hard clauses and maximise the weighted sum of satisfied soft clauses.
- Check the MaxSAT competition

The Example of Graph Coloring: A Possible MaxSAT Model

Let $G = (V, E)$ be an undirected graph where V is the set of vertices and E is the set of edges. In the (decision version of the) graph colouring problem, we are given k colours to colour the graph such that no two adjacent nodes share the same colour.

- Propose a MaxSAT model for the minimisation version of the problem. That is, given $G = (V, E)$, we seek to find the minimum value of k to satisfy the colouring requirements.

The Example of Graph Coloring: A Possible MaxSAT Model

Let $G = (V, E)$ be an undirected graph where V is the set of vertices and E is the set of edges. In the (decision version of the) graph colouring problem, we are given k colours to colour the graph such that no two adjacent nodes share the same colour.

- Propose a MaxSAT model for the minimisation version of the problem. That is, given $G = (V, E)$, we seek to find the minimum value of k to satisfy the colouring requirements.

The Example of Graph Coloring: A Possible MaxSAT Model

- We shall extend the previous model:
- Let u_a be a Boolean variable that is True iff. the colour $a \in [1, k]$ is used
- Consider the previous model $SAT(V, E, k)$ with k an upper bound.
- All the previous clauses are hard.
- Add the following hard clauses:

$$\forall i \in [1, n], \forall a \in [1, k] : \neg u_a \rightarrow \neg x_i^a$$

- Eventually we can add symmetry breaking constraints: $u_a \rightarrow u_{a-1}$
- Then add the soft clauses:

$$\forall a \in [1, k] : \neg u_a$$

- A MaxSAT Optimal solution satisfies all the hard coloring clauses (valid colouring) and maximizes the number of non used colours.

Extensions: Quantified Boolean Formula (QBF)

- A QBF has the form Q.F , where F is a CNF-SAT formulae, and Q is a sequence of quantified variables ($\forall x$ or $\exists x$).
- Example $\forall x, \exists y, \exists z, \ (x \vee \neg y) \wedge (\neg y \vee z)$
- QBF Solver Competition:
  `https://www.qbflib.org/solvers_list.php`

Extensions: Satisfiability Modulo Theories (SMT)

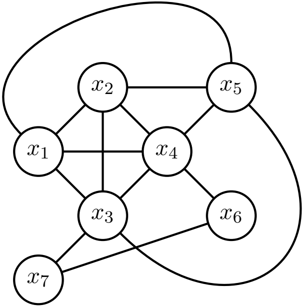
- SMT extends SAT by allowing higher level constraints
- Such constraints belong to certain theories
- Examples of theories include linear integer arithmetic, linear real arithmetic, difference logic, etc
- Check the SAT/SMT summer schools
  `http://satassociation.org/sat-smt-school.html`

Exercise: SAT for Machine Learning

- Let $F = [f_1, \dots f_k]$ be a set of k features and $E = [e_1, \ \dots e_n]$ a set of n examples.
- We want to build an undirected acyclic graph for prediction
- Task1: Propose a model for the topology of the graph
- Task 2: Extend the model to make sure that each example is well classified
- Task 3: Adapt the model to maximize the accuracy of the model

Exercise: Clique

A clique in a graph $G(V, E)$ (where V is the set of vertices and E is the set of edges). A clique in G is a set of vertices $C \subseteq V$ such that $\forall a, b \in C, \{a, b\} \in E$. For examples, in the example below: $\{x_1, x_2, x_3, x_4, x_5\}$ is a clique and $\{x_3, x_6, x_7\}$ is not a clique.



- Propose a SAT model to find a clique of size $\geq k$ for a graph $G(V, E)$.

- Propose a SAT model to find a clique of size $\geq k$ for a graph $G(V, E)$.
- A possible solution:
  - x_i true iff v_i is in the clique
  - For each $\{i, j\} \notin E$:

$$\neg x_i \vee \neg x_j$$

  - Clique size:

$$\sum x_i \geq k$$

  - Implied constraints: If a vertex v_i has less than k edges it shouldn't be part of the clique:

$$\neg x_i$$

MaxSAT

MaxSAT

- Adapt your model into a MaxSAT formulae to find a clique with a maximum size

MaxSAT

- Adapt your model into a MaxSAT formulae to find a clique with a maximum size

  Same model without carnality constraints, without implies constraints, and each x_i is added as a soft clause

Exercise: Shortest Path

Let $G(V, E)$ be a directed graph (where V is the set of vertices and E is the set of directed edges). Suppose that G has a one source $s \in V$ and one sink $o \in V$.
Propose a SAT model to find a path from s to o.
Adapt your model to find a shortest path

Conflict Driven Clause Learning

Modern SAT Solvers: Conflict Driven Clause Learning (CDCL)

Modern SAT Solvers: Conflict Driven Clause Learning (CDCL)

- [Silva and Sakallah, 1999, Moskewicz et al., 2001]

Modern SAT Solvers: Conflict Driven Clause Learning (CDCL)

- [Silva and Sakallah, 1999, Moskewicz et al., 2001]
- DPLL [Davis et al., 1962] $\oplus$ Resolution [Robinson, 1965]

Modern SAT Solvers: Conflict Driven Clause Learning (CDCL)

- [Silva and Sakallah, 1999, Moskewicz et al., 2001]
- DPLL [Davis et al., 1962] $\oplus$ Resolution [Robinson, 1965]
- DPLL: Backtracking + Unit Propagation

Modern SAT Solvers: Conflict Driven Clause Learning (CDCL)

- [Silva and Sakallah, 1999, Moskewicz et al., 2001]
- DPLL [Davis et al., 1962] $\oplus$ Resolution [Robinson, 1965]
- DPLL: Backtracking + Unit Propagation
- Resolution: Learning based on the rule $(l \vee c_1) \wedge (\neg l \vee c_2) \Rightarrow (c_1 \vee c_2)$

Modern SAT Solvers: Conflict Driven Clause Learning (CDCL)

- [Silva and Sakallah, 1999, Moskewicz et al., 2001]
- DPLL [Davis et al., 1962] $\oplus$ Resolution [Robinson, 1965]
- DPLL: Backtracking + Unit Propagation
- Resolution: Learning based on the rule
  $(l \vee c_1) \wedge (\neg l \vee c_2) \Rightarrow (c_1 \vee c_2)$
- **Can be seen as a CP Solver (Search, propagation) augmented by clause learning**

Modern SAT Solvers: Conflict Driven Clause Learning (CDCL)

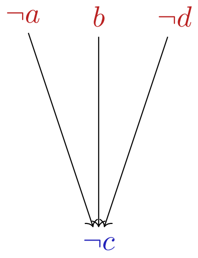
- [Silva and Sakallah, 1999, Moskewicz et al., 2001]
- DPLL [Davis et al., 1962] $\oplus$ Resolution [Robinson, 1965]
- DPLL: Backtracking + Unit Propagation
- Resolution: Learning based on the rule $(l \vee c_1) \wedge (\neg l \vee c_2) \Rightarrow (c_1 \vee c_2)$
- **Can be seen as a CP Solver (Search, propagation) augmented by clause learning**
- But also :
  - Activity-based branching
  - Lazy data structures (2-Watched Literals)
  - Clause Database Reduction
  - Simplifications
  - Restarts
  - ...

Exercise: Propose a filtering algorithm to prune the variables domain in a given clause

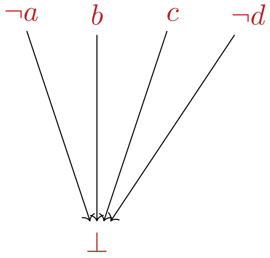
Unit Propagation

Given a clause C of arity n. If $n-1$ literals are false then set the last one to be true.

Example: $(a \vee \neg b \vee \neg c \vee d)$



$\neg a \wedge b \wedge \neg d \Rightarrow \neg c$



$\neg a \wedge b \wedge c \wedge \neg d \Rightarrow \perp$

Algorithm 1: Unit Propagation

Data: A clause C

if *All literals in C are false* **then**
    return Failure ;
else
    if *Only one literal $l \in C$ is unassigned and the rest are false* **then**
        Make l true ;
    end
end

Unit Propagation

- Observe first that propagation happens only in two cases:
  - The clause becomes unit (i.e., all variables except one is instantiated): Propagate the only uninstantiated literal to satisfy the clause
  - All literals are instantiated and none of them satisfy the clause

Unit Propagation

- Observe first that propagation happens only in two cases:
  - The clause becomes unit (i.e., all variables except one is instantiated): Propagate the only uninstantiated literal to satisfy the clause
  - All literals are instantiated and none of them satisfy the clause
- Therefore for each clause C, as long as there are two literals non instantiated in C, nothing happens!

Unit Propagation

- Observe first that propagation happens only in two cases:
  - The clause becomes unit (i.e., all variables except one is instantiated): Propagate the only uninstantiated literal to satisfy the clause
  - All literals are instantiated and none of them satisfy the clause
- Therefore for each clause C, as long as there are two literals non instantiated in C, nothing happens!
- The idea of the Two-watched literals is to keep 2 literals for every clause that are not instantiated. Those literals will "watch the clause" and guarantee that no propagation is needed.

Unit Propagation

- Observe first that propagation happens only in two cases:
  - The clause becomes unit (i.e., all variables except one is instantiated): Propagate the only uninstantiated literal to satisfy the clause
  - All literals are instantiated and none of them satisfy the clause
- Therefore for each clause C, as long as there are two literals non instantiated in C, nothing happens!
- The idea of the Two-watched literals is to keep 2 literals for every clause that are not instantiated. Those literals will "watch the clause" and guarantee that no propagation is needed.
- If a literal watching a clause C becomes $false$, look for replacement. If no replacement found, then perform propagation

Algorithm 2: Two watched Literals (decision d)

```
▷ Assuming initially that all variables are unassigned and that each clause contains at least 2 literals
▷ For each clause C, W[C] is initialized with a set that contains two variables in C
▷ For each variable x, B[x] is the set of clauses watched by x
▷ d is the latest decision ;
S ← {d} ;
while S ≠ ∅ do
    Let x ∈ S ;
    S ← S \ {x} ;
    while B[x] ≠ ∅ do
        Let C ∈ B[x] ;
        if x does not not satisfy C then
            W[C] ← W[C] \ {x} ;
            if ∃x' ∈ C \ W[C] such that x' is unassigned then
                W[C] ← W[C] ∪ {x'} ;
                B[x'] ← B[x'] ∪ {C} ;
            else
                Let y ∈ W[C] ;
                if y is not assigned then
                    assign y to a value that satisfies C ;
                    S ← S ∪ {y} ;
                    S ← ∅
                else
                    if y does not satisfy C then
                        return FAILURE ;
                    end
                end
            end
        end
    end
end
```

Learning and Backjumping

Learning and Backjumping

- Definition: Explaining a failure: $l_1 \wedge \ldots \wedge l_n \rightarrow \perp$ where $\neg l_1 \vee \ldots \vee \neg l_n$ is the clause triggering the failure

Learning and Backjumping

- Definition: Explaining a failure: $l_1 \wedge \ldots \wedge l_n \rightarrow \perp$ where $\neg l_1 \vee \ldots \vee \neg l_n$ is the clause triggering the failure
- Definition: Explaining a propagation of l: $l_1 \wedge \ldots \wedge l_n \rightarrow l$ where $\neg l_1 \vee \ldots \vee \neg l_n \vee \neg l$ is the triggering clause

Learning and Backjumping

- Definition: Explaining a failure: $l_1 \wedge \ldots \wedge l_n \rightarrow \perp$ where $\neg l_1 \vee \ldots \vee \neg l_n$ is the clause triggering the failure
- Definition: Explaining a propagation of l: $l_1 \wedge \ldots \wedge l_n \rightarrow l$ where $\neg l_1 \vee \ldots \vee \neg l_n \vee \neg l$ is the triggering clause
- At each conflict learn a new clause as following:

Learning and Backjumping

- Definition: Explaining a failure: $l_1 \wedge \ldots \wedge l_n \rightarrow \perp$ where $\neg l_1 \vee \ldots \vee \neg l_n$ is the clause triggering the failure
- Definition: Explaining a propagation of l: $l_1 \wedge \ldots \wedge l_n \rightarrow l$ where $\neg l_1 \vee \ldots \vee \neg l_n \vee \neg l$ is the triggering clause
- At each conflict learn a new clause as following:
- Start with the explanation from the clause triggering failure in the form of $l_1 \wedge \ldots \wedge l_n \rightarrow \perp$ and let it be the initial explanation

Learning and Backjumping

- Definition: Explaining a failure: $l_1 \wedge \ldots \wedge l_n \rightarrow \perp$ where $\neg l_1 \vee \ldots \vee \neg l_n$ is the clause triggering the failure
- Definition: Explaining a propagation of l: $l_1 \wedge \ldots \wedge l_n \rightarrow l$ where $\neg l_1 \vee \ldots \vee \neg l_n \vee \neg l$ is the triggering clause
- At each conflict learn a new clause as following:
- Start with the explanation from the clause triggering failure in the form of $l_1 \wedge \ldots \wedge l_n \rightarrow \perp$ and let it be the initial explanation
- While there is more than one literal propagated in the last level in the current explanation, replace it with its explanation from the triggering clause

Learning and Backjumping

- Definition: Explaining a failure: $l_1 \wedge \ldots \wedge l_n \rightarrow \perp$ where $\neg l_1 \vee \ldots \vee \neg l_n$ is the clause triggering the failure
- Definition: Explaining a propagation of l: $l_1 \wedge \ldots \wedge l_n \rightarrow l$ where $\neg l_1 \vee \ldots \vee \neg l_n \vee \neg l$ is the triggering clause
- At each conflict learn a new clause as following:
- Start with the explanation from the clause triggering failure in the form of $l_1 \wedge \ldots \wedge l_n \rightarrow \perp$ and let it be the initial explanation
- While there is more than one literal propagated in the last level in the current explanation, replace it with its explanation from the triggering clause
- When there is only one literal *uip* propagated in the last level in the current explanation, learn the associated new clause C,

Learning and Backjumping

- Definition: Explaining a failure: $l_1 \wedge \ldots \wedge l_n \rightarrow \perp$ where $\neg l_1 \vee \ldots \vee \neg l_n$ is the clause triggering the failure
- Definition: Explaining a propagation of l: $l_1 \wedge \ldots \wedge l_n \rightarrow l$ where $\neg l_1 \vee \ldots \vee \neg l_n \vee \neg l$ is the triggering clause
- At each conflict learn a new clause as following:
- Start with the explanation from the clause triggering failure in the form of $l_1 \wedge \ldots \wedge l_n \rightarrow \perp$ and let it be the initial explanation
- While there is more than one literal propagated in the last level in the current explanation, replace it with its explanation from the triggering clause
- When there is only one literal *uip* propagated in the last level in the current explanation, learn the associated new clause C, backjump (to the last level of propagated literals in C),

Learning and Backjumping

- Definition: Explaining a failure: $l_1 \wedge \ldots \wedge l_n \rightarrow \perp$ where $\neg l_1 \vee \ldots \vee \neg l_n$ is the clause triggering the failure
- Definition: Explaining a propagation of l: $l_1 \wedge \ldots \wedge l_n \rightarrow l$ where $\neg l_1 \vee \ldots \vee \neg l_n \vee \neg l$ is the triggering clause
- At each conflict learn a new clause as following:
- Start with the explanation from the clause triggering failure in the form of $l_1 \wedge \ldots \wedge l_n \rightarrow \perp$ and let it be the initial explanation
- While there is more than one literal propagated in the last level in the current explanation, replace it with its explanation from the triggering clause
- When there is only one literal uip propagated in the last level in the current explanation, learn the associated new clause C, backjump (to the last level of propagated literals in C), propagate $\neg uip$ via the new clause,

Learning and Backjumping

- Definition: Explaining a failure: $l_1 \wedge \ldots \wedge l_n \rightarrow \perp$ where $\neg l_1 \vee \ldots \vee \neg l_n$ is the clause triggering the failure
- Definition: Explaining a propagation of l: $l_1 \wedge \ldots \wedge l_n \rightarrow l$ where $\neg l_1 \vee \ldots \vee \neg l_n \vee \neg l$ is the triggering clause
- At each conflict learn a new clause as following:
- Start with the explanation from the clause triggering failure in the form of $l_1 \wedge \ldots \wedge l_n \rightarrow \perp$ and let it be the initial explanation
- While there is more than one literal propagated in the last level in the current explanation, replace it with its explanation from the triggering clause
- When there is only one literal uip propagated in the last level in the current explanation, learn the associated new clause C, backjump (to the last level of propagated literals in C), propagate $\neg uip$ via the new clause, and continue the exploration

Conflict Analysis

Algorithm 1: 1-UIP-with-Propagators

$\Psi \leftarrow explain(\perp)$;

while $|\{q \in \Psi \mid level(q) = current\ level\}| > 1$ **do**

  $p \leftarrow \arg\max_q(\{rank(q) \mid level(q) = current\ level\ \wedge\ q \in \Psi\})$;

  $\Psi \leftarrow \Psi \cup \{q \mid q \in explain(p) \wedge level(q) > 0\} \setminus \{p\}$;

return Ψ ;

Conflict Analysis

Algorithm 1: 1-UIP-with-Propagators

$\Psi \leftarrow explain(\bot)$;
while $|\{q \in \Psi \mid level(q) = current\ level\}| > 1$ **do**
  $p \leftarrow \arg\max_q(\{rank(q) \mid level(q) = current\ level \ \wedge \ q \in \Psi\})$;
  $\Psi \leftarrow \Psi \cup \{q \mid q \in explain(p) \wedge level(q) > 0\} \setminus \{p\}$;
return Ψ ;

- Why stop with one literal l propagated at the last level ?

Conflict Analysis

Algorithm 1: 1-UIP-with-Propagators

$\Psi \leftarrow explain(\bot)$;
while $|\{q \in \Psi \mid level(q) = current\ level\}| > 1$ **do**
  $p \leftarrow \arg\max_q(\{rank(q) \mid level(q) = current\ level \ \wedge \ q \in \Psi\})$;
  $\Psi \leftarrow \Psi \cup \{q \mid q \in explain(p) \wedge level(q) > 0\} \setminus \{p\}$;
return Ψ ;

- Why stop with one literal l propagated at the last level ?
- **To make sure that when the algorithm backjumps, propagation takes place by making l true**
- When backjumping using a clause that contains more than one literal propagated at the last level, then no propagation can be performed.

Implication Graph

	f		

$\neg a \lor \color{red}{\neg f} \lor g$

$\neg a \lor \neg b \lor \neg h$

$a \lor c$

$a \lor \neg i \lor \neg l$

$a \lor \neg k \lor \neg j$

$b \lor d$

$b \lor g \lor \neg n$

$b \lor \color{red}{\neg f} \lor n \lor k$

$\neg c \lor k$

$\neg c \lor \neg k \lor \neg i \lor l$

$c \lor h \lor n \lor \neg m$

$c \lor l$

$d \lor \neg k \lor l$

$d \lor \neg g \lor l$

$\neg g \lor n \lor o$

$h \lor \neg o \lor \neg j \lor n$

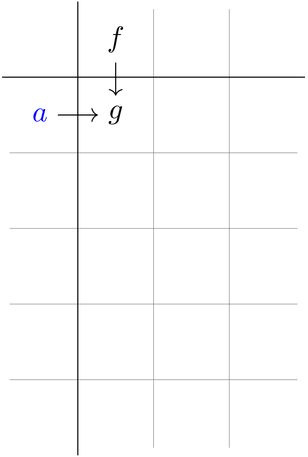
$\neg i \lor j$

$\neg d \lor \neg l \lor \neg m$

$\neg e \lor m \lor \neg n$

$\color{red}{\neg f} \lor h \lor i$

Implication Graph



$\neg a \vee \neg f \vee g$

$\neg a \vee \neg b \vee \neg h$

$a \vee c$

$a \vee \neg i \vee \neg l$

$a \vee \neg k \vee \neg j$

$b \vee d$

$b \vee g \vee \neg n$

$b \vee \neg f \vee n \vee k$

$\neg c \vee k$

$\neg c \vee \neg k \vee \neg i \vee l$

$c \vee h \vee n \vee \neg m$

$c \vee l$

$d \vee \neg k \vee l$

$d \vee \neg g \vee l$

$\neg g \vee n \vee o$

$h \vee \neg o \vee \neg j \vee n$

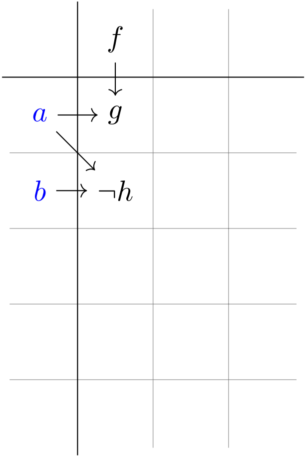
$\neg i \vee j$

$\neg d \vee \neg l \vee \neg m$

$\neg e \vee m \vee \neg n$

$\neg f \vee h \vee i$

Implication Graph



$\neg a \vee \neg f \vee g$

$\neg a \vee \neg b \vee \neg h$

$a \vee c$

$a \vee \neg i \vee \neg l$

$a \vee \neg k \vee \neg j$

$b \vee d$

$b \vee g \vee \neg n$

$b \vee \neg f \vee n \vee k$

$\neg c \vee k$

$\neg c \vee \neg k \vee \neg i \vee l$

$c \vee h \vee n \vee \neg m$

$c \vee l$

$d \vee \neg k \vee l$

$d \vee \neg g \vee l$

$\neg g \vee n \vee o$

$h \vee \neg o \vee \neg j \vee n$

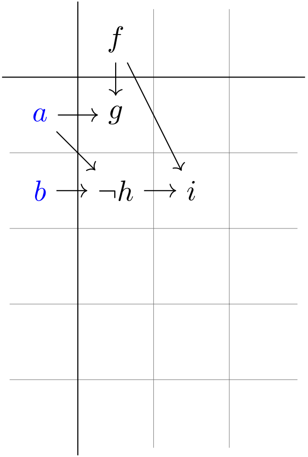
$\neg i \vee j$

$\neg d \vee \neg l \vee \neg m$

$\neg e \vee m \vee \neg n$

$\neg f \vee h \vee i$

Implication Graph



$\neg a \lor \neg f \lor g$

$\neg a \lor \neg b \lor \neg h$

$a \lor c$

$a \lor \neg i \lor \neg l$

$a \lor \neg k \lor \neg j$

$b \lor d$

$b \lor g \lor \neg n$

$b \lor \neg f \lor n \lor k$

$\neg c \lor k$

$\neg c \lor \neg k \lor \neg i \lor l$

$c \lor h \lor n \lor \neg m$

$c \lor l$

$d \lor \neg k \lor l$

$d \lor \neg g \lor l$

$\neg g \lor n \lor o$

$h \lor \neg o \lor \neg j \lor n$

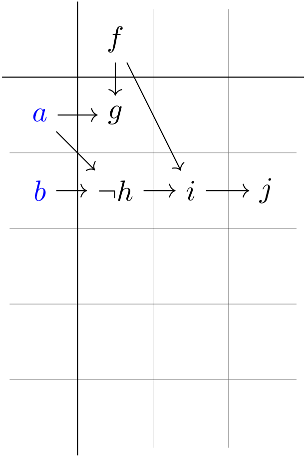
$\neg i \lor j$

$\neg d \lor \neg l \lor \neg m$

$\neg e \lor m \lor \neg n$

$\neg f \lor h \lor i$

Implication Graph



$\neg a \vee \neg f \vee g$

$\neg a \vee \neg b \vee \neg h$

$a \vee c$

$a \vee \neg i \vee \neg l$

$a \vee \neg k \vee \neg j$

$b \vee d$

$b \vee g \vee \neg n$

$b \vee \neg f \vee n \vee k$

$\neg c \vee k$

$\neg c \vee \neg k \vee \neg i \vee l$

$c \vee h \vee n \vee \neg m$

$c \vee l$

$d \vee \neg k \vee l$

$d \vee \neg g \vee l$

$\neg g \vee n \vee o$

$h \vee \neg o \vee \neg j \vee n$

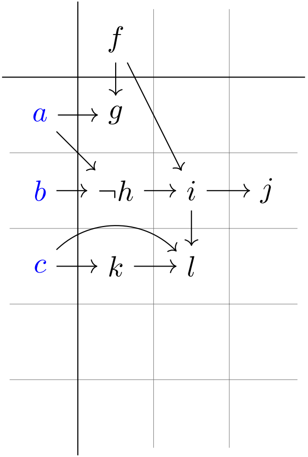
$\neg i \vee j$

$\neg d \vee \neg l \vee \neg m$

$\neg e \vee m \vee \neg n$

$\neg f \vee h \vee i$

Implication Graph



$\neg a \vee \neg f \vee g$

$\neg a \vee \neg b \vee \neg h$

$a \vee c$

$a \vee \neg i \vee \neg l$

$a \vee \neg k \vee \neg j$

$b \vee d$

$b \vee g \vee \neg n$

$b \vee \neg f \vee n \vee k$

$\neg c \vee k$

$\neg c \vee \neg k \vee \neg i \vee l$

$c \vee h \vee n \vee \neg m$

$c \vee l$

$d \vee \neg k \vee l$

$d \vee \neg g \vee l$

$\neg g \vee n \vee o$

$h \vee \neg o \vee \neg j \vee n$

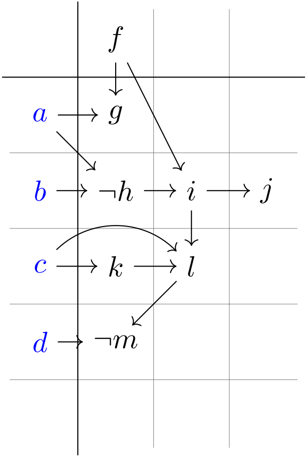
$\neg i \vee j$

$\neg d \vee \neg l \vee \neg m$

$\neg e \vee m \vee \neg n$

$\neg f \vee h \vee i$

Implication Graph



$\neg a \vee \neg f \vee g$

$\neg a \vee \neg b \vee \neg h$

$a \vee c$

$a \vee \neg i \vee \neg l$

$a \vee \neg k \vee \neg j$

$b \vee d$

$b \vee g \vee \neg n$

$b \vee \neg f \vee n \vee k$

$\neg c \vee k$

$\neg c \vee \neg k \vee \neg i \vee l$

$c \vee h \vee n \vee \neg m$

$c \vee l$

$d \vee \neg k \vee l$

$d \vee \neg g \vee l$

$\neg g \vee n \vee o$

$h \vee \neg o \vee \neg j \vee n$

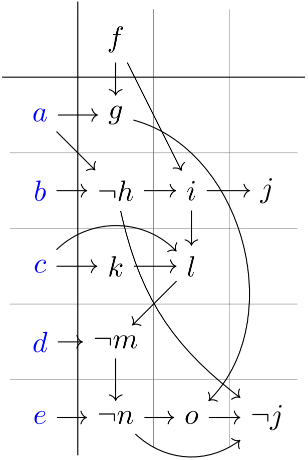
$\neg i \vee j$

$\neg d \vee \neg l \vee \neg m$

$\neg e \vee m \vee \neg n$

$\neg f \vee h \vee i$

Implication Graph



$\neg a \vee \neg f \vee g$

$\neg a \vee \neg b \vee \neg h$

$a \vee c$

$a \vee \neg i \vee \neg l$

$a \vee \neg k \vee \neg j$

$b \vee d$

$b \vee g \vee \neg n$

$b \vee \neg f \vee n \vee k$

$\neg c \vee k$

$\neg c \vee \neg k \vee \neg i \vee l$

$c \vee h \vee n \vee \neg m$

$c \vee l$

$d \vee \neg k \vee l$

$d \vee \neg g \vee l$

$\neg g \vee n \vee o$

$h \vee \neg o \vee \neg j \vee n$

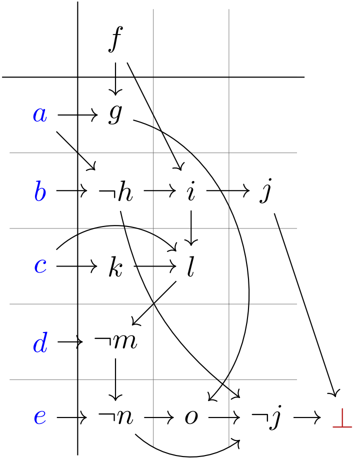
$\neg i \vee j$

$\neg d \vee \neg l \vee \neg m$

$\neg e \vee m \vee \neg n$

$\neg f \vee h \vee i$

Implication Graph



$\neg a \lor \neg f \lor g$

$\neg a \lor \neg b \lor \neg h$

$a \lor c$

$a \lor \neg i \lor \neg l$

$a \lor \neg k \lor \neg j$

$b \lor d$

$b \lor g \lor \neg n$

$b \lor \neg f \lor n \lor k$

$\neg c \lor k$

$\neg c \lor \neg k \lor \neg i \lor l$

$c \lor h \lor n \lor \neg m$

$c \lor l$

$d \lor \neg k \lor l$

$d \lor \neg g \lor l$

$\neg g \lor n \lor o$

$\boxed{h \lor \neg o \lor \neg j \lor n}$

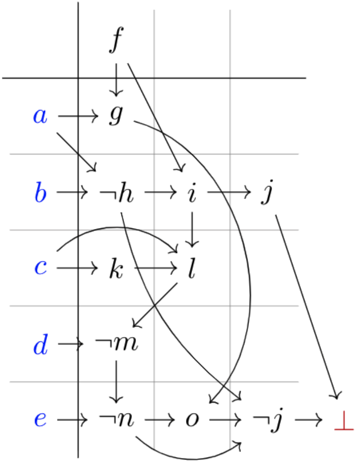
$\neg i \lor j$

$\neg d \lor \neg l \lor \neg m$

$\neg e \lor m \lor \neg n$

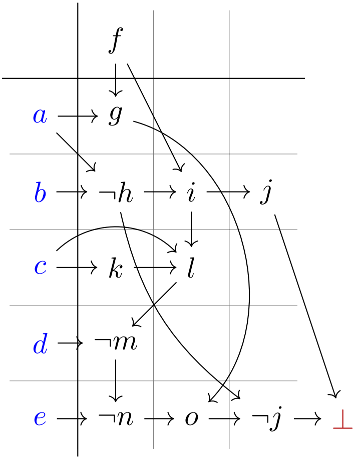
$\neg f \lor h \lor i$

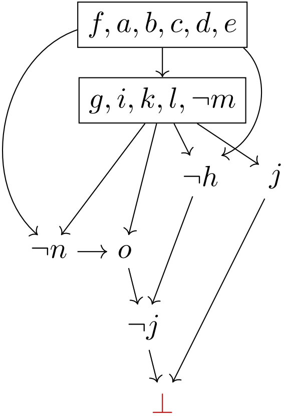
Implication Graph



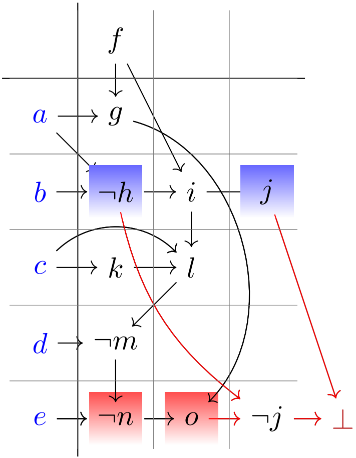
$\neg a \vee \neg f \vee g$	$c \vee h \vee n \vee \neg m$
$\neg a \vee \neg b \vee \neg h$	$c \vee l$
$a \vee c$	$d \vee \neg k \vee l$
$a \vee \neg i \vee \neg l$	$d \vee \neg g \vee l$
$a \vee \neg k \vee \neg j$	$\neg g \vee n \vee o$
$b \vee d$	$\boxed{h \vee \neg o \vee \neg j \vee n}$
$b \vee g \vee \neg n$	$\neg i \vee j$
$b \vee \neg f \vee n \vee k$	$\neg d \vee \neg l \vee \neg m$
$\neg c \vee k$	$\neg e \vee m \vee \neg n$
$\neg c \vee \neg k \vee \neg i \vee l$	$\neg f \vee h \vee i$

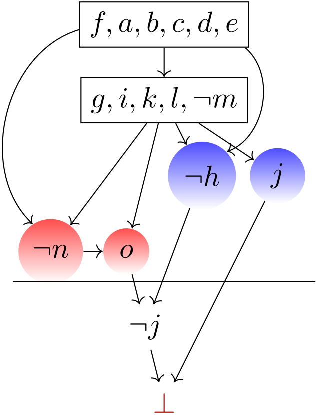
Conflict Analysis



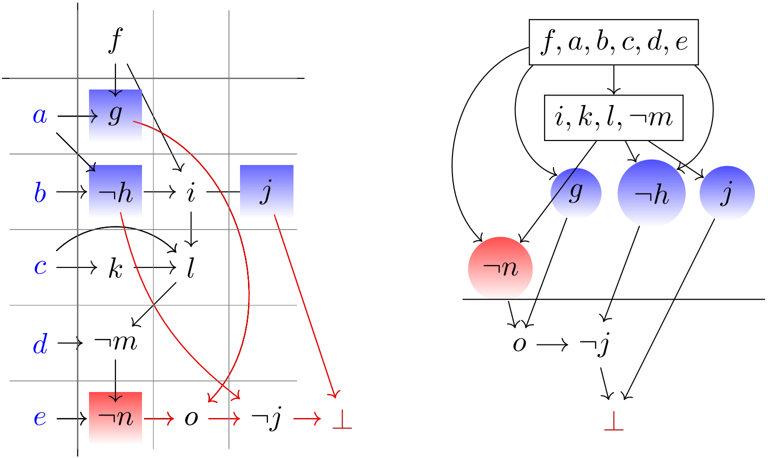


Conflict Analysis

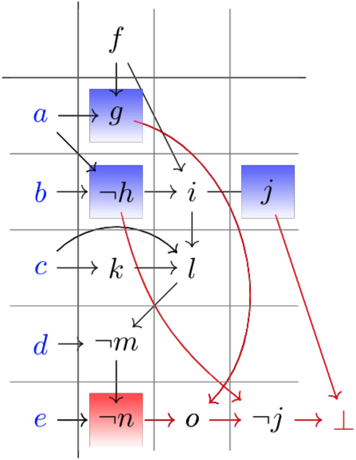


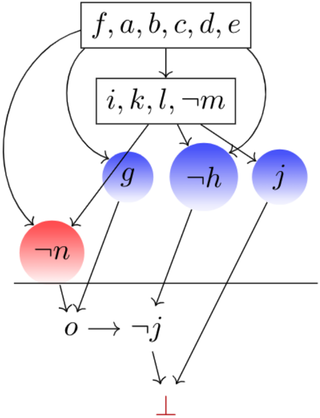


Conflict Analysis

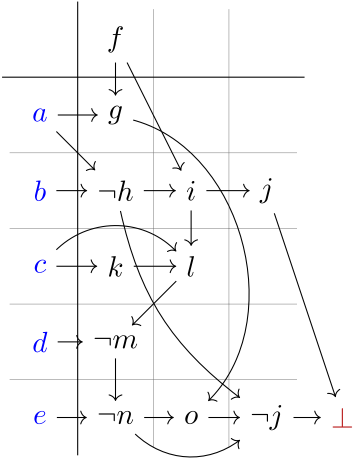


Conflict Analysis



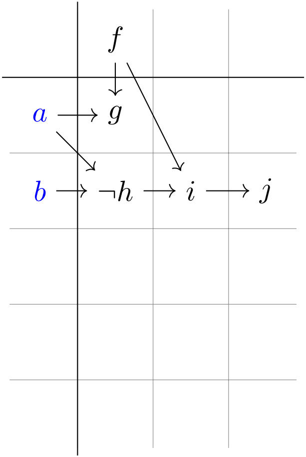


Conflict analysis



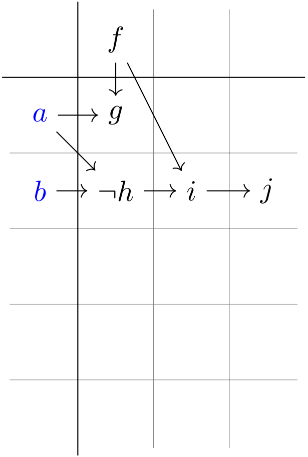
$\neg a \vee \neg f \vee g$	$c \vee h \vee n \vee \neg m$
$\neg a \vee \neg b \vee \neg h$	$c \vee l$
$a \vee c$	$d \vee \neg k \vee l$
$a \vee \neg i \vee \neg l$	$d \vee \neg g \vee l$
$a \vee \neg k \vee \neg j$	$\neg g \vee n \vee o$
$b \vee d$	$h \vee \neg o \vee \neg j \vee n$
$b \vee g \vee \neg n$	$\neg i \vee j$
$b \vee \neg f \vee n \vee k$	$\neg d \vee \neg l \vee \neg m$
$\neg c \vee k$	$\neg e \vee m \vee \neg n$
$\neg c \vee \neg k \vee \neg i \vee l$	$\neg f \vee h \vee i$

Conflict analysis



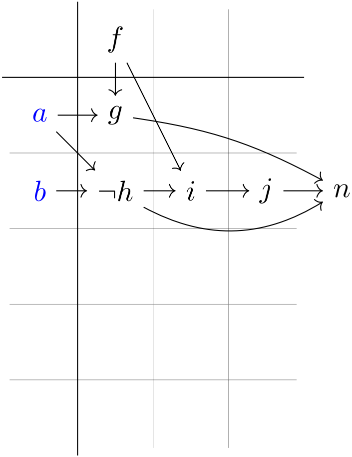
$\neg a \vee \neg f \vee g$	$c \vee h \vee n \vee \neg m$
$\neg a \vee \neg b \vee \neg h$	$c \vee l$
$a \vee c$	$d \vee \neg k \vee l$
$a \vee \neg i \vee \neg l$	$d \vee \neg g \vee l$
$a \vee \neg k \vee \neg j$	$\neg g \vee n \vee o$
$b \vee d$	$h \vee \neg o \vee \neg j \vee n$
$b \vee g \vee \neg n$	$\neg i \vee j$
$b \vee \neg f \vee n \vee k$	$\neg d \vee \neg l \vee \neg m$
$\neg c \vee k$	$\neg e \vee m \vee \neg n$
$\neg c \vee \neg k \vee \neg i \vee l$	$\neg f \vee h \vee i$

Conflict analysis



$\neg a \vee \neg f \vee g$
$\neg a \vee \neg b \vee \neg h$
$a \vee c$
$a \vee \neg i \vee \neg l$
$a \vee \neg k \vee \neg j$
$b \vee d$
$b \vee g \vee \neg n$
$b \vee \neg f \vee n \vee k$
$\neg c \vee k$
$\neg c \vee \neg k \vee \neg i \vee l$
$c \vee h \vee n \vee \neg m$
$c \vee l$
$d \vee \neg k \vee l$
$d \vee \neg g \vee l$
$\neg g \vee n \vee o$
$h \vee \neg o \vee \neg j \vee n$
$\neg i \vee j$
$\neg d \vee \neg l \vee \neg m$
$\neg e \vee m \vee \neg n$
$\neg f \vee h \vee i$
$\boxed{\neg g \vee h \vee \neg j \vee n}$

Conflict analysis



$\neg a \lor \neg f \lor g$
$\neg a \lor \neg b \lor \neg h$
$a \lor c$
$a \lor \neg i \lor \neg l$
$a \lor \neg k \lor \neg j$
$b \lor d$
$b \lor g \lor \neg n$
$b \lor \neg f \lor n \lor k$
$\neg c \lor k$
$\neg c \lor \neg k \lor \neg i \lor l$

$c \lor h \lor n \lor \neg m$
$c \lor l$
$d \lor \neg k \lor l$
$d \lor \neg g \lor l$
$\neg g \lor n \lor o$
$h \lor \neg o \lor \neg j \lor n$
$\neg i \lor j$
$\neg d \lor \neg l \lor \neg m$
$\neg e \lor m \lor \neg n$
$\neg f \lor h \lor i$
$\boxed{\neg g \lor h \lor \neg j \lor n}$

Boosting Search through Randomization and Restarts [Gomes et al., 1998]

Boosting Search through Randomization and Restarts [Gomes et al., 1998]

Heavy-tail phenomena (SAT and CP)

At any time during the experiment there is a non-negligible probability of hitting a problem that requires exponentially more time to solve than any that has been encountered before.

Boosting Search through Randomization and Restarts [Gomes et al., 1998]

Heavy-tail phenomena (SAT and CP)

At any time during the experiment there is a non-negligible probability of hitting a problem that requires exponentially more time to solve than any that has been encountered before.

Hardness = Instance $\oplus$ deterministic algorithm.

Boosting Search through Randomization and Restarts [Gomes et al., 1998]

Heavy-tail phenomena (SAT and CP)

At any time during the experiment there is a non-negligible probability of hitting a problem that requires exponentially more time to solve than any that has been encountered before.

Hardness = Instance $\oplus$ deterministic algorithm.

- Randomization: breaking ties, random decision between k best choices, ...
- Restarts: Geometric/Luby

Other techniques

Other techniques

- Forgetting clauses: The number of the learnt clauses can be exponential, we sometimes need to free some space by forgetting some clauses.

Other techniques

- Forgetting clauses: The number of the learnt clauses can be exponential, we sometimes need to free some space by forgetting some clauses.
- VSIDS (Variable State Independent Decaying Sum): VSIDS is a popular variable ordering heuristic that is based on the notion of activity.

Other techniques

- Forgetting clauses: The number of the learnt clauses can be exponential, we sometimes need to free some space by forgetting some clauses.
- VSIDS (Variable State Independent Decaying Sum): VSIDS is a popular variable ordering heuristic that is based on the notion of activity. The activity of a variable is measured by the number of times it participates in the conflict analysis.

Other techniques

- Forgetting clauses: The number of the learnt clauses can be exponential, we sometimes need to free some space by forgetting some clauses.
- VSIDS (Variable State Independent Decaying Sum): VSIDS is a popular variable ordering heuristic that is based on the notion of activity. The activity of a variable is measured by the number of times it participates in the conflict analysis. Each time a variable x is used during conflict analysis, its activity is incremented.

Other techniques

- Forgetting clauses: The number of the learnt clauses can be exponential, we sometimes need to free some space by forgetting some clauses.
- VSIDS (Variable State Independent Decaying Sum): VSIDS is a popular variable ordering heuristic that is based on the notion of activity. The activity of a variable is measured by the number of times it participates in the conflict analysis. Each time a variable x is used during conflict analysis, its activity is incremented. From time to time, the counters are divided by a constant (to diminish the effect of early conflicts).

SAT Solvers (Few examples)

- MiniSat: http://minisat.se/
- Glucose: `http://www.labri.fr/perso/lsimon/glucose/`
- LingeLing `http://fmv.jku.at/lingeling`
- Any Solver by Armin Biere `http://fmv.jku.at/software/index.html`
- Any winner from past and future SAT competitions: https://www.satcompetition.org/

SAT vs CSP

Back to Constraint Programming

Back to Constraint Programming

- A constraint is a finite relation (i.e., a subset of a Cartesian product)

Back to Constraint Programming

- A constraint is a finite relation (i.e., a subset of a Cartesian product)
- A constraint can be expressed in extension (table constraint) or intention (expression)

Back to Constraint Programming

- A constraint is a finite relation (i.e., a subset of a Cartesian product)
- A constraint can be expressed in extension (table constraint) or intention (expression)
- A constraint network is defined by a triplet $P = (X, D, C)$ where
  - X is a set of variables
  - D is a set of domains for the variables in X
  - C is a set of constraints

Back to Constraint Programming

- A constraint is a finite relation (i.e., a subset of a Cartesian product)
- A constraint can be expressed in extension (table constraint) or intention (expression)
- A constraint network is defined by a triplet $P = (X, D, C)$ where
  - X is a set of variables
  - D is a set of domains for the variables in X
  - C is a set of constraints
- The constraint satisfaction problem (CSP) is the problem of deciding if a constraint network has a solution

Back to Constraint Programming

- A constraint is a finite relation (i.e., a subset of a Cartesian product)
- A constraint can be expressed in extension (table constraint) or intention (expression)
- A constraint network is defined by a triplet $P = (X, D, C)$ where
  - X is a set of variables
  - D is a set of domains for the variables in X
  - C is a set of constraints
- The constraint satisfaction problem (CSP) is the problem of deciding if a constraint network has a solution
- Mostly solvable by backtracking algorithms (Search and Filtering)

Search

Search

Search

Search

- Search: decisions to explore the search tree

Search

Search

- Search: decisions to explore the search tree
- Search in CP= variable ordering + value ordering

Search

Search

- Search: decisions to explore the search tree
- Search in CP= variable ordering + value ordering
- Standard or customized

Search

Search

- Search: decisions to explore the search tree
- Search in CP= variable ordering + value ordering
- Standard or customized

Variable Ordering

'Fail-first' principle [Haralick and Elliott, 1980]:
"To succeed, try first where you are most likely to fail"

Search

Search

- Search: decisions to explore the search tree
- Search in CP= variable ordering + value ordering
- Standard or customized

Variable Ordering

'Fail-first' principle [Haralick and Elliott, 1980]:
"To succeed, try first where you are most likely to fail"

Value Ordering

'Succeed-first' [Geelen, 1992]:
"Follow the best chances leading to a solution"

Filtering

- Filtering (propagation/pruning): inferences based on the current state

Filtering

- Filtering (propagation/pruning): inferences based on the current state
- Constraint $\leftrightarrow$ a propagator

Filtering

- Filtering (propagation/pruning): inferences based on the current state
- Constraint $\leftrightarrow$ a propagator
- Propagators are executed sequentially before making any decision

Filtering

- Filtering (propagation/pruning): inferences based on the current state
- Constraint $\leftrightarrow$ a propagator
- Propagators are executed sequentially before making any decision
- The level of pruning $\leftrightarrow$ local consistency (for instance, bound consistency, arc consistency, etc)

Filtering

- Filtering (propagation/pruning): inferences based on the current state
- Constraint $\leftrightarrow$ a propagator
- Propagators are executed sequentially before making any decision
- The level of pruning $\leftrightarrow$ local consistency (for instance, bound consistency, arc consistency, etc)

Arc Consistency

Let C be a constraint and D be a list of domains for the variables in the scope of C.

Filtering

- Filtering (propagation/pruning): inferences based on the current state
- Constraint $\leftrightarrow$ a propagator
- Propagators are executed sequentially before making any decision
- The level of pruning $\leftrightarrow$ local consistency (for instance, bound consistency, arc consistency, etc)

Arc Consistency

Let C be a constraint and D be a list of domains for the variables in the scope of C.
C is Arc Consistent (AC) iff for every variable x in the scope of C, for every value $v \in D(x)$, there exists an assignment w in D satisfying C in which v is assigned to x

Filtering algorithm

Filtering algorithm

- A Filtering algorithm associated with a constraint C takes as input a list of domains (for the variables in the scope of C) and returns a list of domains that are smaller or identical to the original domains.

Filtering algorithm

- A Filtering algorithm associated with a constraint C takes as input a list of domains (for the variables in the scope of C) and returns a list of domains that are smaller or identical to the original domains.
- For a filtering algorithm to be correct: no consistent value should be removed (by consistent we mean to belong to a satisfying assignment).

Filtering algorithm

- A Filtering algorithm associated with a constraint C takes as input a list of domains (for the variables in the scope of C) and returns a list of domains that are smaller or identical to the original domains.
- For a filtering algorithm to be correct: no consistent value should be removed (by consistent we mean to belong to a satisfying assignment).
- If each domain is a singleton, the propagator must be able to check if the assignment corresponds to a solution or not.

CP vs. SAT

CP vs. SAT

- CP: rich modelling language, powerful filtering, dedicated search strategies

CP vs. SAT

- CP: rich modelling language, powerful filtering, dedicated search strategies
- SAT: simple input format, clause learning and backjumping, autonomous search

CP vs. SAT

- CP: rich modelling language, powerful filtering, dedicated search strategies
- SAT: simple input format, clause learning and backjumping, autonomous search

- Every CSP can be encoded into SAT

CP vs. SAT

- CP: rich modelling language, powerful filtering, dedicated search strategies
- SAT: simple input format, clause learning and backjumping, autonomous search

- Every CSP can be encoded into SAT
- When should we encode to SAT, when shouldn't we?

CP vs. SAT

- CP: rich modelling language, powerful filtering, dedicated search strategies
- SAT: simple input format, clause learning and backjumping, autonomous search

- Every CSP can be encoded into SAT
- When should we encode to SAT, when shouldn't we?
- CP vs. SAT: a fundamental difference is the presence of global reasoning in CP and clause learning in SAT

CP vs. SAT : To decompose or not to decompose?

CP vs. SAT : To decompose or not to decompose?

- Decomposition is the task of reformulating a (global) constraint into smaller and simpler constraints.

CP vs. SAT : To decompose or not to decompose?

- Decomposition is the task of reformulating a (global) constraint into smaller and simpler constraints.
- Take the example of AllDifferent: it can be decomposed into simple binary inequalities. **Remember the tutorial!**.

CP vs. SAT : To decompose or not to decompose?

- Decomposition is the task of reformulating a (global) constraint into smaller and simpler constraints.
- Take the example of AllDifferent: it can be decomposed into simple binary inequalities. **Remember the tutorial!**.
- **In general, decomposition makes the filtering weaker. We lose the powerful filtering from the global constraints by decomposing.**
- On the one hand, by decomposing into clauses, we loose the powerful filtering from CP

CP vs. SAT : To decompose or not to decompose?

- Decomposition is the task of reformulating a (global) constraint into smaller and simpler constraints.
- Take the example of AllDifferent: it can be decomposed into simple binary inequalities. **Remember the tutorial!**.
- **In general, decomposition makes the filtering weaker. We lose the powerful filtering from the global constraints by decomposing.**
- On the one hand, by decomposing into clauses, we loose the powerful filtering from CP
- Also the size of the encoding matters. An exponential encoding is better avoided!

CP vs. SAT : To decompose or not to decompose?

- Decomposition is the task of reformulating a (global) constraint into smaller and simpler constraints.
- Take the example of AllDifferent: it can be decomposed into simple binary inequalities. **Remember the tutorial!**.
- **In general, decomposition makes the filtering weaker. We lose the powerful filtering from the global constraints by decomposing.**
- On the one hand, by decomposing into clauses, we loose the powerful filtering from CP
- Also the size of the encoding matters. An exponential encoding is better avoided!
- On the other hand, clause learning in SAT is quite powerful to learn new clauses and to backjump in the search tree

CP vs. SAT : To decompose or not to decompose?

- Decomposition is the task of reformulating a (global) constraint into smaller and simpler constraints.
- Take the example of AllDifferent: it can be decomposed into simple binary inequalities. **Remember the tutorial!**.
- **In general, decomposition makes the filtering weaker. We lose the powerful filtering from the global constraints by decomposing.**
- On the one hand, by decomposing into clauses, we loose the powerful filtering from CP
- Also the size of the encoding matters. An exponential encoding is better avoided!
- On the other hand, clause learning in SAT is quite powerful to learn new clauses and to backjump in the search tree
- **Can we find something that takes advantage from both worlds?**

CP vs. SAT : To decompose or not to decompose?

- Decomposition is the task of reformulating a (global) constraint into smaller and simpler constraints.
- Take the example of AllDifferent: it can be decomposed into simple binary inequalities. **Remember the tutorial!**.
- **In general, decomposition makes the filtering weaker. We lose the powerful filtering from the global constraints by decomposing.**
- On the one hand, by decomposing into clauses, we loose the powerful filtering from CP
- Also the size of the encoding matters. An exponential encoding is better avoided!
- On the other hand, clause learning in SAT is quite powerful to learn new clauses and to backjump in the search tree
- **Can we find something that takes advantage from both worlds?** $\rightarrow$ **Clause learning in CP**

Modern Constraint Solvers: Hybrid CP/SAT

Modern Constraint Solvers: Hybrid CP/SAT

- Learning from conflict

Modern Constraint Solvers: Hybrid CP/SAT

- Learning from conflict
- Based on the notion of explanation

Modern Constraint Solvers: Hybrid CP/SAT

- Learning from conflict
- Based on the notion of explanation
- Generalized Nogoods[Katsirelos and Bacchus, 2005], Lazy Clause generation [Ohrimenko et al., 2009], Clause Learning in sequencing and scheduling problems [Siala, 2015], ...

Modern Constraint Solvers: Hybrid CP/SAT

- Learning from conflict
- Based on the notion of explanation
- Generalized Nogoods[Katsirelos and Bacchus, 2005], Lazy Clause generation [Ohrimenko et al., 2009], Clause Learning in sequencing and scheduling problems [Siala, 2015], ...

Learning in CP

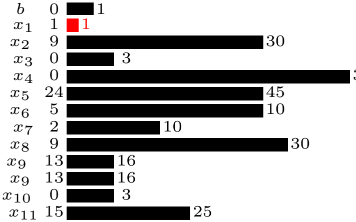
$$
\begin{aligned}
& x_1 + x_7 \geq 4 \land \\
& x_2 + x_{10} \geq 11 \land \\
& x_3 + x_9 = 16 \land \\
& x_5 \geq x_8 + x_9 \land \\
& b \leftrightarrow (x_9 - x_4 = 14) \land \\
& b \rightarrow (x_6 \geq 7) \land \\
& b \rightarrow (x_6 + x_7 \leq 9) \land \\
& x_{11} \geq x_9 + x_{10}
\end{aligned}
$$

b	0	1
x_1	1	30
x_2	9	30
x_3	0	3
x_4	0	30
x_5	24	45
x_6	5	10
x_7	2	10
x_8	9	30
x_9	13	16
x_9	13	16
x_{10}	0	3
x_{11}	15	25

Learning in CP

$x_1 + x_7 \geq 4 \wedge$
$x_2 + x_{10} \geq 11 \wedge$
$x_3 + x_9 = 16 \wedge$
$x_5 \geq x_8 + x_9 \wedge$
$b \leftrightarrow (x_9 - x_4 = 14) \wedge$
$b \rightarrow (x_6 \geq 7) \wedge$
$b \rightarrow (x_6 + x_7 \leq 9) \wedge$
$x_{11} \geq x_9 + x_{10}$

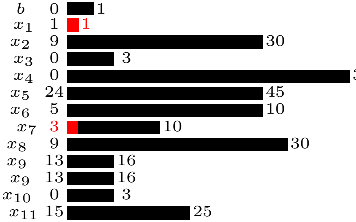
$[\![x_1 = 1]\!]$



Learning in CP

$$
\begin{aligned}
& x_1 + x_7 \geq 4 \wedge \\
& x_2 + x_{10} \geq 11 \wedge \\
& x_3 + x_9 = 16 \wedge \\
& x_5 \geq x_8 + x_9 \wedge \\
& b \leftrightarrow (x_9 - x_4 = 14) \wedge \\
& b \rightarrow (x_6 \geq 7) \wedge \\
& b \rightarrow (x_6 + x_7 \leq 9) \wedge \\
& x_{11} \geq x_9 + x_{10}
\end{aligned}
$$

$[\![x_1 = 1]\!] \rightarrow [\![x_7 \geq 3]\!]$

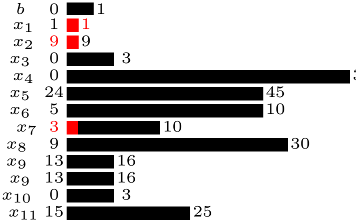


Learning in CP

$$
\begin{aligned}
&x_1 + x_7 \geq 4 \wedge \\
&x_2 + x_{10} \geq 11 \wedge \\
&x_3 + x_9 = 16 \wedge \\
&x_5 \geq x_8 + x_9 \wedge \\
&b \leftrightarrow (x_9 - x_4 = 14) \wedge \\
&b \rightarrow (x_6 \geq 7) \wedge \\
&b \rightarrow (x_6 + x_7 \leq 9) \wedge \\
&x_{11} \geq x_9 + x_{10}
\end{aligned}
$$

$[\![x_1 = 1]\!] \rightarrow [\![x_7 \geq 3]\!]$

$[\![x_2 = 9]\!]$



Learning in CP

$[\![x_1 = 1]\!] \rightarrow [\![x_7 \geq 3]\!]$

$[\![x_2 = 9]\!] \rightarrow [\![x_{10} \geq 2]\!]$

$$
\begin{aligned}
& x_1 + x_7 \geq 4 \wedge \\
& x_2 + x_{10} \geq 11 \wedge \\
& x_3 + x_9 = 16 \wedge \\
& x_5 \geq x_8 + x_9 \wedge \\
& b \leftrightarrow (x_9 - x_4 = 14) \wedge \\
& b \rightarrow (x_6 \geq 7) \wedge \\
& b \rightarrow (x_6 + x_7 \leq 9) \wedge \\
& x_{11} \geq x_9 + x_{10}
\end{aligned}
$$

b	0	1
x_1	1	1
x_2	9	9
x_3	0	3
x_4	0	[illegible]
x_5	24	45
x_6	5	10
x_7	3	10
x_8	9	30
x_9	13	16
x_9	13	16
x_{10}	2	3
x_{11}	15	25

Learning in CP

$[\![x_1 = 1]\!] \rightarrow [\![x_7 \geq 3]\!]$

$[\![x_2 = 9]\!] \rightarrow [\![x_{10} \geq 2]\!]$

$[\![x_3 = 2]\!]$

$x_1 + x_7 \geq 4 \wedge$
$x_2 + x_{10} \geq 11 \wedge$
$x_3 + x_9 = 16 \wedge$
$x_5 \geq x_8 + x_9 \wedge$
$b \leftrightarrow (x_9 - x_4 = 14) \wedge$
$b \rightarrow (x_6 \geq 7) \wedge$
$b \rightarrow (x_6 + x_7 \leq 9) \wedge$
$x_{11} \geq x_9 + x_{10}$

b	0	1
x_1	1	1
x_2	9	9
x_3	2	2
x_4	0	3
x_5	24	45
x_6	5	10
x_7	3	10
x_8	9	30
x_9	13	16
x_9	13	16
x_{10}	2	3
x_{11}	15	25

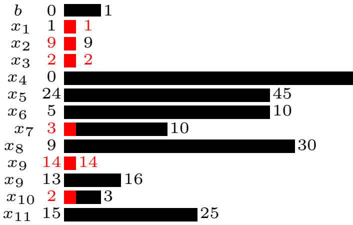
Learning in CP

$[\![x_1 = 1]\!] \rightarrow [\![x_7 \geq 3]\!]$

$[\![x_2 = 9]\!] \rightarrow [\![x_{10} \geq 2]\!]$

$[\![x_3 = 2]\!] \rightarrow [\![x_9 = 14]\!]$

$$
\begin{aligned}
&x_1 + x_7 \geq 4 \wedge \\
&x_2 + x_{10} \geq 11 \wedge \\
&x_3 + x_9 = 16 \wedge \\
&x_5 \geq x_8 + x_9 \wedge \\
&b \leftrightarrow (x_9 - x_4 = 14) \wedge \\
&b \rightarrow (x_6 \geq 7) \wedge \\
&b \rightarrow (x_6 + x_7 \leq 9) \wedge \\
&x_{11} \geq x_9 + x_{10}
\end{aligned}
$$



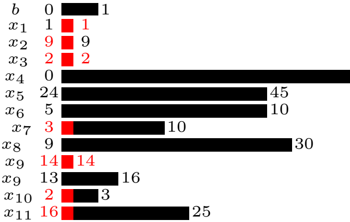
Learning in CP

$[\![x_1 = 1]\!] \rightarrow [\![x_7 \geq 3]\!]$

$[\![x_2 = 9]\!] \rightarrow [\![x_{10} \geq 2]\!]$

$[\![x_3 = 2]\!] \rightarrow [\![x_9 = 14]\!] \rightarrow [\![x_{11} \geq 16]\!]$

$$
\begin{aligned}
& x_1 + x_7 \geq 4 \wedge \\
& x_2 + x_{10} \geq 11 \wedge \\
& x_3 + x_9 = 16 \wedge \\
& x_5 \geq x_8 + x_9 \wedge \\
& b \leftrightarrow (x_9 - x_4 = 14) \wedge \\
& b \rightarrow (x_6 \geq 7) \wedge \\
& b \rightarrow (x_6 + x_7 \leq 9) \wedge \\
& x_{11} \geq x_9 + x_{10}
\end{aligned}
$$



Learning in CP

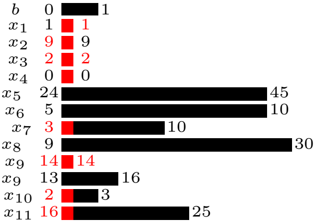
$[\![x_1 = 1]\!] \rightarrow [\![x_7 \geq 3]\!]$

$[\![x_2 = 9]\!] \rightarrow [\![x_{10} \geq 2]\!]$

$[\![x_3 = 2]\!] \rightarrow [\![x_9 = 14]\!] \rightarrow [\![x_{11} \geq 16]\!]$

$[\![x_4 = 0]\!]$

$x_1 + x_7 \geq 4 \wedge$
$x_2 + x_{10} \geq 11 \wedge$
$x_3 + x_9 = 16 \wedge$
$x_5 \geq x_8 + x_9 \wedge$
$b \leftrightarrow (x_9 - x_4 = 14) \wedge$
$b \rightarrow (x_6 \geq 7) \wedge$
$b \rightarrow (x_6 + x_7 \leq 9) \wedge$
$x_{11} \geq x_9 + x_{10}$



Learning in CP

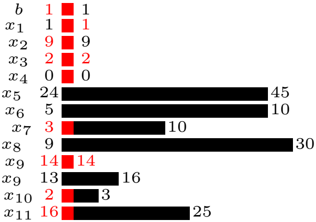
$[\![x_1 = 1]\!] \rightarrow [\![x_7 \geq 3]\!]$

$[\![x_2 = 9]\!] \rightarrow [\![x_{10} \geq 2]\!]$

$[\![x_3 = 2]\!] \rightarrow [\![x_9 = 14]\!] \rightarrow [\![x_{11} \geq 16]\!]$

$[\![x_4 = 0]\!] \longrightarrow [\![b = 1]\!]$

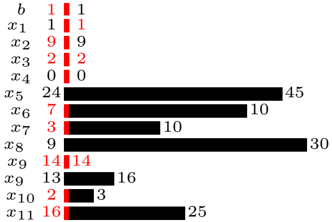
$x_1 + x_7 \geq 4 \wedge$
$x_2 + x_{10} \geq 11 \wedge$
$x_3 + x_9 = 16 \wedge$
$x_5 \geq x_8 + x_9 \wedge$
$b \leftrightarrow (x_9 - x_4 = 14) \wedge$
$b \rightarrow (x_6 \geq 7) \wedge$
$b \rightarrow (x_6 + x_7 \leq 9) \wedge$
$x_{11} \geq x_9 + x_{10}$



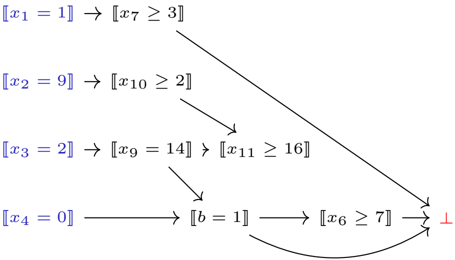
Learning in CP

$$
\begin{tikzcd}
\color{blue}{[\![x_1 = 1]\!]} \arrow[r] & {[\![x_7 \geq 3]\!]} & & \\
\color{blue}{[\![x_2 = 9]\!]} \arrow[r] & {[\![x_{10} \geq 2]\!]} \arrow[dr] & & \\
\color{blue}{[\![x_3 = 2]\!]} \arrow[r] & {[\![x_9 = 14]\!]} \arrow[r] \arrow[dr] & {[\![x_{11} \geq 16]\!]} & \\
\color{blue}{[\![x_4 = 0]\!]} \arrow[rr] & & {[\![b = 1]\!]} \arrow[r] & {[\![x_6 \geq 7]\!]}
\end{tikzcd}
$$

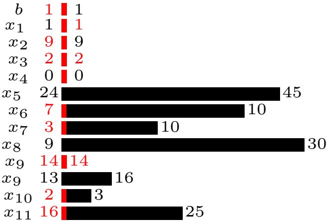
$x_1 + x_7 \geq 4 \wedge$
$x_2 + x_{10} \geq 11 \wedge$
$x_3 + x_9 = 16 \wedge$
$x_5 \geq x_8 + x_9 \wedge$
$b \leftrightarrow (x_9 - x_4 = 14) \wedge$
$b \rightarrow (x_6 \geq 7) \wedge$
$b \rightarrow (x_6 + x_7 \leq 9) \wedge$
$x_{11} \geq x_9 + x_{10}$



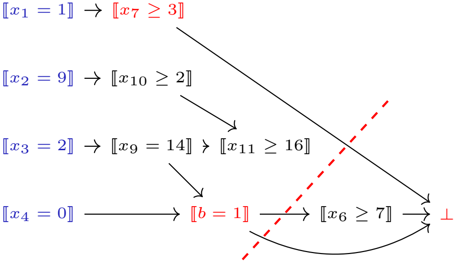
Learning in CP



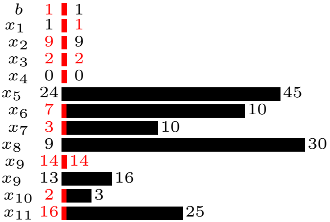
$$
\begin{aligned}
&x_1 + x_7 \geq 4 \wedge \\
&x_2 + x_{10} \geq 11 \wedge \\
&x_3 + x_9 = 16 \wedge \\
&x_5 \geq x_8 + x_9 \wedge \\
&b \leftrightarrow (x_9 - x_4 = 14) \wedge \\
&b \rightarrow (x_6 \geq 7) \wedge \\
&b \rightarrow (x_6 + x_7 \leq 9) \wedge \\
&x_{11} \geq x_9 + x_{10}
\end{aligned}
$$



Learning in CP

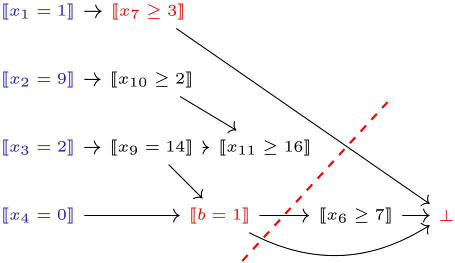


$x_1 + x_7 \geq 4 \wedge$
$x_2 + x_{10} \geq 11 \wedge$
$x_3 + x_9 = 16 \wedge$
$x_5 \geq x_8 + x_9 \wedge$
$b \leftrightarrow (x_9 - x_4 = 14) \wedge$
$b \rightarrow (x_6 \geq 7) \wedge$
$b \rightarrow (x_6 + x_7 \leq 9) \wedge$
$x_{11} \geq x_9 + x_{10}$

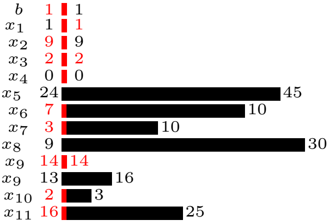


- Conflict analysis: $[\![b = 1]\!] \wedge [\![x_7 \geq 3]\!] \Rightarrow \perp$

Learning in CP



$x_1 + x_7 \geq 4 \wedge$
$x_2 + x_{10} \geq 11 \wedge$
$x_3 + x_9 = 16 \wedge$
$x_5 \geq x_8 + x_9 \wedge$
$b \leftrightarrow (x_9 - x_4 = 14) \wedge$
$b \rightarrow (x_6 \geq 7) \wedge$
$b \rightarrow (x_6 + x_7 \leq 9) \wedge$
$x_{11} \geq x_9 + x_{10}$



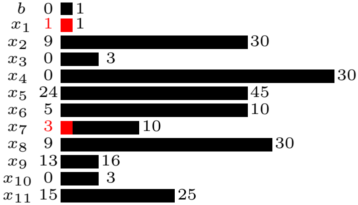
- Conflict analysis: $[\![b = 1]\!] \wedge [\![x_7 \geq 3]\!] \Rightarrow \perp$
- New clause: $[\![b \neq 1]\!] \vee [\![x_7 \leq 2]\!]$

Learning in CP

$[\![x_1 = 1]\!] \rightarrow [\![x_7 \geq 3]\!]$

- Conflict analysis: $[\![b = 1]\!] \wedge [\![x_7 \geq 3]\!] \Rightarrow \perp$
- New clause: $[\![b \neq 1]\!] \vee [\![x_7 \leq 2]\!]$
- Backtrack to level 1

$$\begin{array}{l} x_1 + x_7 \geq 4 \wedge \\ x_2 + x_{10} \geq 11 \wedge \\ x_3 + x_9 = 16 \wedge \\ x_5 \geq x_8 + x_9 \wedge \\ b \leftrightarrow (x_9 - x_4 = 14) \wedge \\ b \rightarrow (x_6 \geq 7) \wedge \\ b \rightarrow (x_6 + x_7 \leq 9) \wedge \\ x_{11} \geq x_9 + x_{10} \end{array}$$

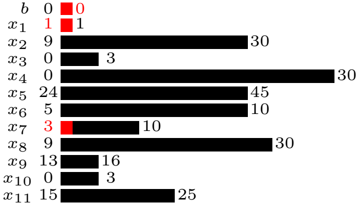


Learning in CP

$$\textcolor{blue}{[\![x_1 = 1]\!]} \rightarrow [\![x_7 \geq 3]\!] \longrightarrow \textcolor{red}{[\![b = 0]\!]}$$

- Conflict analysis: $[\![b = 1]\!] \wedge [\![x_7 \geq 3]\!] \Rightarrow \perp$
- New clause: $[\![b \neq 1]\!] \vee [\![x_7 \leq 2]\!]$
- Backtrack to level 1
- Propagate the learnt clause

$x_1 + x_7 \geq 4 \wedge$
$x_2 + x_{10} \geq 11 \wedge$
$x_3 + x_9 = 16 \wedge$
$x_5 \geq x_8 + x_9 \wedge$
$b \leftrightarrow (x_9 - x_4 = 14) \wedge$
$b \rightarrow (x_6 \geq 7) \wedge$
$b \rightarrow (x_6 + x_7 \leq 9) \wedge$
$x_{11} \geq x_9 + x_{10}$

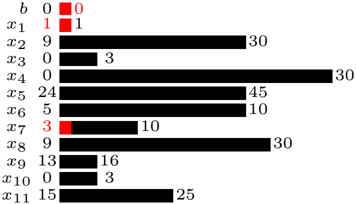


Learning in CP

$[\![x_1 = 1]\!] \rightarrow [\![x_7 \geq 3]\!] \longrightarrow [\![b = 0]\!]$

- Conflict analysis: $[\![b = 1]\!] \wedge [\![x_7 \geq 3]\!] \Rightarrow \bot$
- New clause: $[\![b \neq 1]\!] \vee [\![x_7 \leq 2]\!]$
- Backtrack to level 1
- Propagate the learnt clause
- Continue exploration

$$
\begin{aligned}
& x_1 + x_7 \geq 4 \wedge \\
& x_2 + x_{10} \geq 11 \wedge \\
& x_3 + x_9 = 16 \wedge \\
& x_5 \geq x_8 + x_9 \wedge \\
& b \leftrightarrow (x_9 - x_4 = 14) \wedge \\
& b \rightarrow (x_6 \geq 7) \wedge \\
& b \rightarrow (x_6 + x_7 \leq 9) \wedge \\
& x_{11} \geq x_9 + x_{10}
\end{aligned}
$$



Conflict analysis

Algorithm 1: 1-UIP-with-Propagators

$\Psi \leftarrow explain(\perp)$;

while $|\{q \in \Psi \mid level(q) = current\ level\}| > 1$ **do**

  $p \leftarrow \arg\max_q(\{rank(q) \mid level(q) = current\ level\ \wedge\ q \in \Psi\})$;

  $\Psi \leftarrow \Psi \cup \{q \mid q \in explain(p) \wedge level(q) > 0\} \setminus \{p\}$;

return Ψ ;

Explaining constraints

Explaining constraints

- To enable clause learning in CP, each propagator must be able to explain its filtering in the form of clauses ("Lazy Clause Generation").

Explaining constraints

- To enable clause learning in CP, each propagator must be able to explain its filtering in the form of clauses ("Lazy Clause Generation").
- We distinguish two types of explanations:

Explaining constraints

- To enable clause learning in CP, each propagator must be able to explain its filtering in the form of clauses ("Lazy Clause Generation").
- We distinguish two types of explanations:
  - Explaining Failure
  - Explaining Domain filtering

Explaining constraints

- To enable clause learning in CP, each propagator must be able to explain its filtering in the form of clauses ("Lazy Clause Generation").
- We distinguish two types of explanations:
  - Explaining Failure
  - Explaining Domain filtering
- Example: Explain the constraint $X \leq Y$ with two scenarios (failure and propagation).

- Let $(x_1, \ldots, x_n)$ be a sequence of Boolean variables, and let d be a positive integer.
- The CARDINALITY$(x_1, \ldots, x_n, d)$ constraint holds iff exactly d variables from the sequence $(x_1, \ldots, x_n)$ are true.
- Write a filtering algorithm for CARDINALITY.
- What is the time complexity?
- Does it enforce arc consistency?
- Explain the CARDINALITY filtering.

Correction

Algorithm 4: CARDINALITY($[x_1, \ldots, x_n], d$)

if $\left|\{x_j \mid \mathcal{D}(x_j) = \{1\}\}\right| > d$ **then**
  $\mathcal{D} \leftarrow \perp$;
if $\left|\{x_j \mid \mathcal{D}(x_j) = \{0\}\}\right| > n - d$ **then**
  $\mathcal{D} \leftarrow \perp$;
if $\left|\{x_j \mid \mathcal{D}(x_j) = \{1\}\}\right| = d$ **then**
  foreach $i \in \{1..n\}$ **do**
    if $\mathcal{D}(x_i) = \{0, 1\}$ **then**
      $\mathcal{D}(x_i) \leftarrow \{0\}$;
else
  if $\left|\{x_j \mid \mathcal{D}(x_j) = \{0\}\}\right| = n - d$ **then**
    foreach $i \in \{1..n\}$ **do**
      if $\mathcal{D}(x_i) = \{0, 1\}$ **then**
        $\mathcal{D}(x_i) \leftarrow \{1\}$;
return $\mathcal{D}$;

Explaining The Cardinality Constraint

Explaining The Cardinality Constraint

- Failure 1:

$$x^1 \wedge x^2 \wedge \ldots \wedge x^{d+1} \rightarrow \bot$$

Where $D(x^i) = \{1\}$

Explaining The Cardinality Constraint

- Failure 1:
$$x^1 \wedge x^2 \wedge \ldots \wedge x^{d+1} \rightarrow \perp$$
Where $D(x^i) = \{1\}$
- Failure 2:
$$\neg x^1 \wedge \neg x^2 \wedge \neg x^{n-d+1} \rightarrow \perp$$
Where $D(x^i) = \{0\}$
- Explaining the propagating of the value 1: the conjunction of all the assigned variables

Explaining The Cardinality Constraint

- Failure 1:
$$x^1 \wedge x^2 \wedge \ldots \wedge x^{d+1} \rightarrow \bot$$
Where $D(x^i) = \{1\}$
- Failure 2:
$$\neg x^1 \wedge \neg x^2 \wedge \neg x^{n-d+1} \rightarrow \bot$$
Where $D(x^i) = \{0\}$
- Explaining the propagating of the value 1: the conjunction of all the assigned variables
- Explaining the propagating of the value 0: the conjunction of all the assigned variables

Encoding CSP into SAT

- How to encode the variables' domain ?
- How to encode each constraint into a set of clauses ?

Domain Encoding: Quadratic Encoding

- Suppose that $D(x) = \{v_1, \ldots, v_n\}$

Domain Encoding: Quadratic Encoding

- Suppose that $D(x) = \{v_1, \ldots, v_n\}$
- Let x_i be a Boolean variable that is true if $x == x_i$

Domain Encoding: Quadratic Encoding

- Suppose that $D(x) = \{v_1, \ldots, v_n\}$
- Let x_i be a Boolean variable that is true if $x == x_i$
- $x_1 \vee \ldots \vee x_n$

Domain Encoding: Quadratic Encoding

- Suppose that $D(x) = \{v_1, \ldots, v_n\}$
- Let x_i be a Boolean variable that is true if $x == x_i$
- $x_1 \vee \ldots \vee x_n$
- For each $1 \leq i < j \leq n$, encode $x_i \neq x_j$

Domain Encoding: Quadratic Encoding

- Suppose that $D(x) = \{v_1, \ldots, v_n\}$
- Let x_i be a Boolean variable that is true if $x == x_i$
- $x_1 \vee \ldots \vee x_n$
- For each $1 \leq i < j \leq n$, encode $x_i \neq x_j$
- That is, $x_i \rightarrow \neg x_j$

Domain Encoding: Quadratic Encoding

- Suppose that $D(x) = \{v_1, \ldots, v_n\}$
- Let x_i be a Boolean variable that is true if $x == x_i$
- $x_1 \vee \ldots \vee x_n$
- For each $1 \leq i < j \leq n$, encode $x_i \neq x_j$
- That is, $x_i \rightarrow \neg x_j$
- As a clause: $\neg x_i \vee \neg x_j$

Domain Encoding: Quadratic Encoding

- Suppose that $D(x) = \{v_1, \ldots, v_n\}$
- Let x_i be a Boolean variable that is true if $x == x_i$
- $x_1 \vee \ldots \vee x_n$
- For each $1 \leq i < j \leq n$, encode $x_i \neq x_j$
- That is, $x_i \rightarrow \neg x_j$
- As a clause: $\neg x_i \vee \neg x_j$
- The number of variables is linear

Domain Encoding: Quadratic Encoding

- Suppose that $D(x) = \{v_1, \ldots, v_n\}$
- Let x_i be a Boolean variable that is true if $x == x_i$
- $x_1 \vee \ldots \vee x_n$
- For each $1 \leq i < j \leq n$, encode $x_i \neq x_j$
- That is, $x_i \rightarrow \neg x_j$
- As a clause: $\neg x_i \vee \neg x_j$
- The number of variables is linear
- The number of clauses is quadratic

Domain Encoding: Linear Encoding

- Suppose that $D(x) = \{1, \ldots, n\}$

Domain Encoding: Linear Encoding

- Suppose that $D(x) = \{1, \ldots, n\}$
- Let x_i be a Boolean variable that is true if $x == i$

Domain Encoding: Linear Encoding

- Suppose that $D(x) = \{1, \ldots, n\}$
- Let x_i be a Boolean variable that is true if $x == i$
- Let y_j be a Boolean variable that is true if $x \leq j$ where $j \in [1, \ldots n]$

Domain Encoding: Linear Encoding

- Suppose that $D(x) = \{1, \ldots, n\}$
- Let x_i be a Boolean variable that is true if $x == i$
- Let y_j be a Boolean variable that is true if $x \leq j$ where $j \in [1, \ldots n]$
- $x_1 \vee \ldots \vee x_n$

Domain Encoding: Linear Encoding

- Suppose that $D(x) = \{1, \ldots, n\}$
- Let x_i be a Boolean variable that is true if $x == i$
- Let y_j be a Boolean variable that is true if $x \leq j$ where $j \in [1, \ldots n]$
- $x_1 \vee \ldots \vee x_n$
- $y_j \rightarrow y_{j+1}$

Domain Encoding: Linear Encoding

- Suppose that $D(x) = \{1, \ldots, n\}$
- Let x_i be a Boolean variable that is true if $x == i$
- Let y_j be a Boolean variable that is true if $x \leq j$ where $j \in [1, \ldots n]$
- $x_1 \vee \ldots \vee x_n$
- $y_j \rightarrow y_{j+1}$
- $x_i \rightarrow y_i \wedge \neg y_{i-1}$

Domain Encoding: Linear Encoding

- Suppose that $D(x) = \{1, \ldots, n\}$
- Let x_i be a Boolean variable that is true if $x == i$
- Let y_j be a Boolean variable that is true if $x \leq j$ where $j \in [1, \ldots n]$
- $x_1 \vee \ldots \vee x_n$
- $y_j \rightarrow y_{j+1}$
- $x_i \rightarrow y_i \wedge \neg y_{i-1}$
- The number of variables is linear in the size of the domain

Domain Encoding: Linear Encoding

- Suppose that $D(x) = \{1, \ldots, n\}$
- Let x_i be a Boolean variable that is true if $x == i$
- Let y_j be a Boolean variable that is true if $x \leq j$ where $j \in [1, \ldots n]$
- $x_1 \vee \ldots \vee x_n$
- $y_j \rightarrow y_{j+1}$
- $x_i \rightarrow y_i \wedge \neg y_{i-1}$
- The number of variables is linear in the size of the domain
- The number of clauses is linear. However, some clauses are of arity three

Exercise: Constraint encoding ?

- How to encode the AllDifferent constraint ?
- How to encode $\sum_i X_i \leq k$ (X_i is an integer variable)?
- How to encode $\sum_i a_i \times X_i \leq k$?

Take Away Message

Take Away Message

- SAT, CP, MIP, (also, MaxSAT, SMT, QBF, ASP, Pseudo-Boolean) are efficient tools to solve hard combinatorial problems

Take Away Message

- SAT, CP, MIP, (also, MaxSAT, SMT, QBF, ASP, Pseudo-Boolean) are efficient tools to solve hard combinatorial problems
- When you master one or few techniques, it opens the door to work on diverse problems. The more you apply to different problems, the more you learn

Take Away Message

- SAT, CP, MIP, (also, MaxSAT, SMT, QBF, ASP, Pseudo-Boolean) are efficient tools to solve hard combinatorial problems
- When you master one or few techniques, it opens the door to work on diverse problems. The more you apply to different problems, the more you learn
- The choice depends on the problem at hand (is it easy to linearise? what is the size of the SAT encoding? Can we use/invent global constraints?, etc)

Take Away Message

- SAT, CP, MIP, (also, MaxSAT, SMT, QBF, ASP, Pseudo-Boolean) are efficient tools to solve hard combinatorial problems
- When you master one or few techniques, it opens the door to work on diverse problems. The more you apply to different problems, the more you learn
- The choice depends on the problem at hand (is it easy to linearise? what is the size of the SAT encoding? Can we use/invent global constraints?, etc)
- You don't need to implement a solver: use existing ones! check the different solver competitions

Take Away Message

- SAT, CP, MIP, (also, MaxSAT, SMT, QBF, ASP, Pseudo-Boolean) are efficient tools to solve hard combinatorial problems
- When you master one or few techniques, it opens the door to work on diverse problems. The more you apply to different problems, the more you learn
- The choice depends on the problem at hand (is it easy to linearise? what is the size of the SAT encoding? Can we use/invent global constraints?, etc)
- You don't need to implement a solver: use existing ones! check the different solver competitions
- Hybrid approaches are the future: take advantage of diverse methodologies

Take Away Message

- SAT, CP, MIP, (also, MaxSAT, SMT, QBF, ASP, Pseudo-Boolean) are efficient tools to solve hard combinatorial problems
- When you master one or few techniques, it opens the door to work on diverse problems. The more you apply to different problems, the more you learn
- The choice depends on the problem at hand (is it easy to linearise? what is the size of the SAT encoding? Can we use/invent global constraints?, etc)
- You don't need to implement a solver: use existing ones! check the different solver competitions
- Hybrid approaches are the future: take advantage of diverse methodologies

References I

Davis, M., Logemann, G., and Loveland, D. (1962).
A Machine Program for Theorem-proving.
Communications of the ACM, 5(7):394–397.

Gomes, C. P., Selman, B., and Kautz, H. (1998).
Boosting Combinatorial Search Through Randomization.
In *Proceedings of the 15th National Conference on Artificial Intelligence, AAAI'98, and the 10th Conference on Innovative Applications of Artificial Intelligence, IAAI'98, Madison, Wisconsin*, pages 431–437.

Katsirelos, G. and Bacchus, F. (2005).
Generalized NoGoods in CSPs.
In *Proceedings of the 20th National Conference on Artificial Intelligence, AAAI'05, and the 17th Conference on Innovative Applications of Artificial Intelligence, IAAI'05, Pittsburgh, Pennsylvania, USA*, pages 390–396.

Moskewicz, M. W., Madigan, C. F., Zhao, Y., Zhang, L., and Malik, S. (2001).
Chaff: Engineering an Efficient SAT Solver.
In *Proceedings of the 38th Annual Design Automation Conference, DAC'01, Las Vegas, Nevada, USA*, pages 530–535.

References II

Ohrimenko, O., Stuckey, P. J., and Codish, M. (2009).
Propagation via Lazy Clause Generation.
Constraints, 14(3):357–391.

Robinson, J. A. (1965).
A Machine-Oriented Logic Based on the Resolution Principle.
Journal of the ACM, 12(1):23–41.

Siala, M. (2015).
Search, propagation, and learning in sequencing and scheduling problems. (Recherche, propagation et apprentissage dans les problèmes de séquencement et d'ordonnancement).
PhD thesis, INSA Toulouse, France.

Silva, J. a. P. M. and Sakallah, K. A. (1999).
Grasp: a search algorithm for propositional satisfiability.
Computers, IEEE Transactions on, 48(5):506–521.