The Why, How, and What of SAT

袁至誠
Chih-cheng Rex Yuan

Shenzhen University

Friday 21st April, 2023

P and NP

P The class of problems that are "easy":

- $\mathcal{O}(1)$
- $\mathcal{O}(n)$
- $\mathcal{O}(n^3 + k)$

NP The class of problems that are "hard":

- $\mathcal{O}(n!)$
- $\mathcal{O}(2^n)$
- $\mathcal{O}(2^{n^3+k})$

For all intents and purposes, P $\neq$ NP.

Hamiltonian path

A Hamiltonian path of a graph is a path which visits every node of the graph exactly once.
A Hamiltonian path of G is (a, b, d, c).

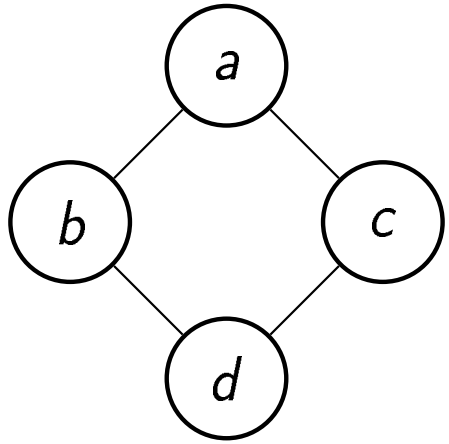
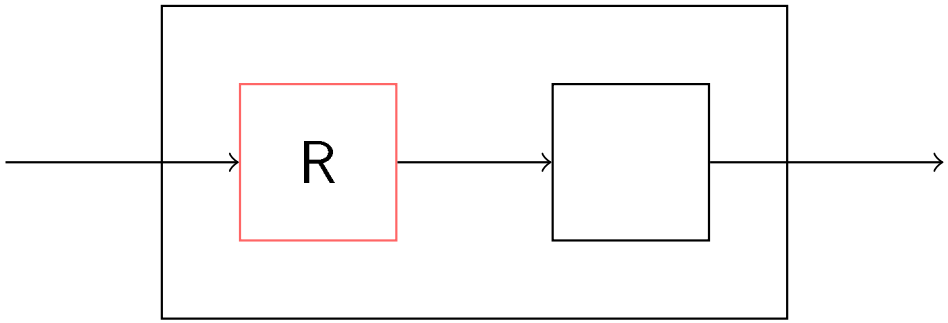


Figure: Undirected $G(\{a, b, c, d\}, E)$

A Hamiltonian path can be found in $\mathcal{O}(n \times n!)$ by enumerating over all possible sequence and checking each of them if it is a path.

Hardness, Completeness, and Reduction

Reduction A reduction is a problem "wrapper". We say A reduces to B if there's an "easy enough" reduction R such that we can solve A by passing the input through R to B.



Hardness If every problem of a class C reduces to a problem A, we say that A is C-hard.
Completeness If A is C-hard and A is also in the class C, we say that A is C-complete.

A C-complete problem can be treated as the "hardest" problem in that class.

Why SAT?

Fact

Hamiltonian path is NP-complete.

Solving Hamiltonian path is tough.

Fact

SAT is NP-complete.

Solving SAT is also tough, but since it's NP-complete, if we can solve it fairly fast, we can solve any NP problem fairly fast.

However, decades of research in SAT-solving has yielded heuristics and implementations that are "unreasonably effective" against large problems (millions of clauses and variables).

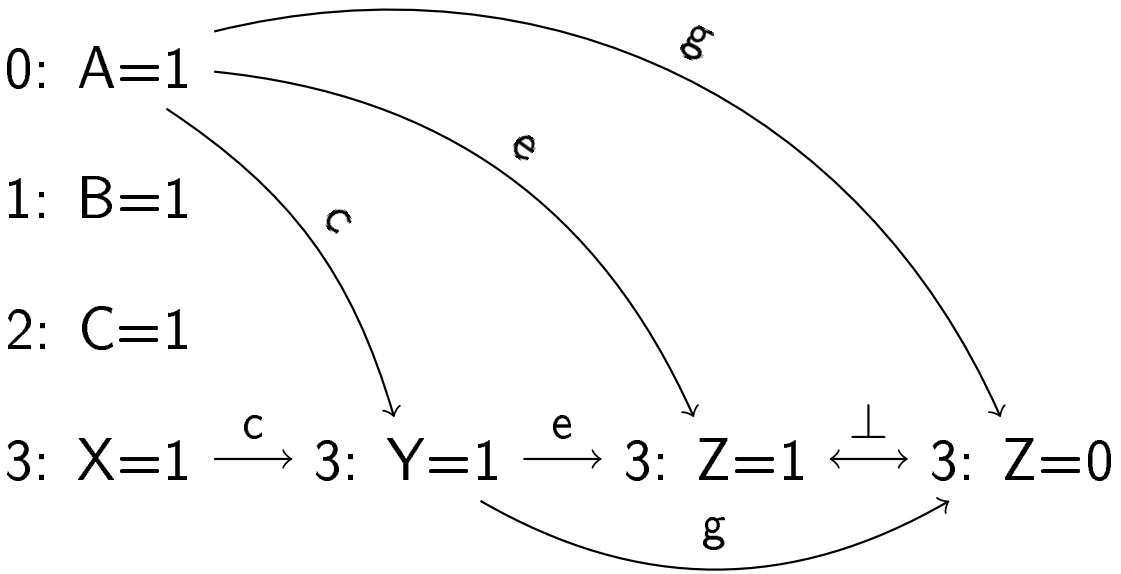
Reduction of Hamiltonian path to SAT

$G = (S, E)$. Set $\mathtt{X}_{i,j}$ to mean the ith position in the Hamiltonian path is occupied by node j.

- Each node j must appear in the path: $\bigwedge_{j \in S} \bigvee_{i \in S} \mathtt{X}_{i,j}$
- No node j appears twice in the path: $\bigwedge_{i,j,k \in S, i \neq k} \neg(\mathtt{X}_{i,j} \wedge \mathtt{X}_{k,j})$
- Every position i on the path must be occupied: $\bigwedge_{i \in S} \bigvee_{j \in S} \mathtt{X}_{i,j}$
- No two nodes j and k occupy the same position in the path: $\bigwedge_{i,j,k \in S, j \neq k} \neg(\mathtt{X}_{i,j} \wedge \mathtt{X}_{i,k})$
- Nonadjacent nodes i and j cannot be adjacent in the path: $\bigwedge_{k \in \{1,\ldots,n-1\}, (i,j) \notin E} \neg(\mathtt{X}_{k,i} \wedge \mathtt{X}_{k+1,j})$

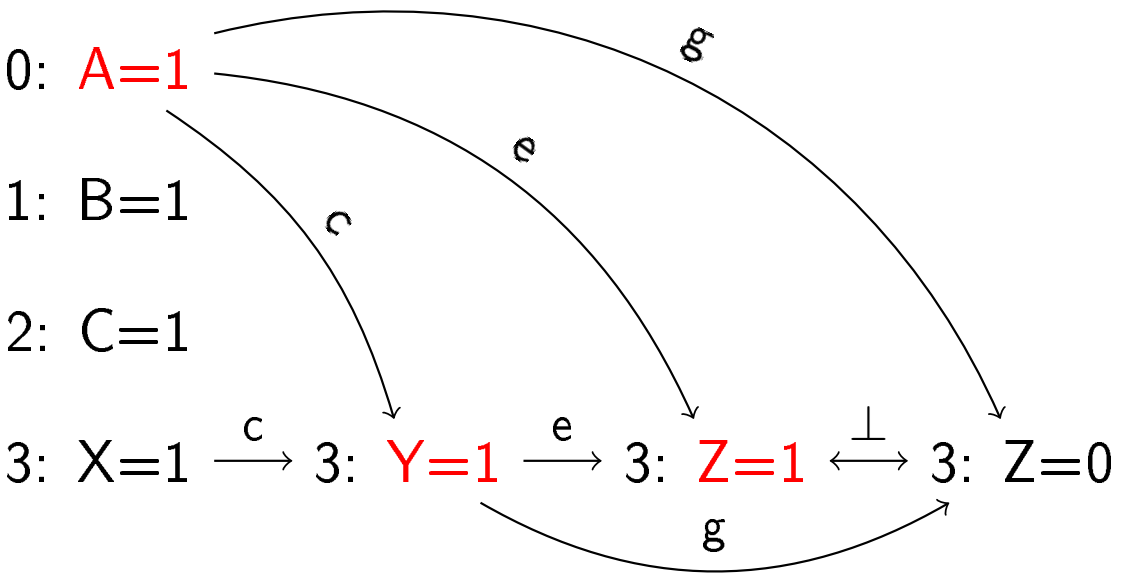
How SAT?

The backbone of modern SAT-solving is the idea of unit propagration and Conflict-Driven Clause-Learning (CDCL). Consider this example from Carsten Sinz and Tomáš Balyo of Karlsruhe Institute of Technology.



$$
\begin{array}{ll}
\{\{A, B\}, & a \\
\{B, C\}, & b \\
\{\neg A, \neg X, Y\}, & c \\
\{\neg A, X, Z\}, & d \\
\{\neg A, \neg Y, Z\}, & e \\
\{\neg A, X, \neg Z\}, & f \\
\{\neg A, \neg Y, \neg Z\}\} & g
\end{array}
$$

CDCL



$$
\begin{array}{ll}
\{\{A, B\}, & a \\
\{B, C\}, & b \\
\{\neg A, \neg X, Y\}, & c \\
\{\neg A, X, Z\}, & d \\
\{\neg A, \neg Y, Z\}, & e \\
\{\neg A, X, \neg Z\}, & f \\
\{\neg A, \neg Y, \neg Z\}\} & g
\end{array}
$$

Conflicting clause: $\neg(A \wedge Y \wedge Z) = \{\neg A, \neg Y, \neg Z\}$

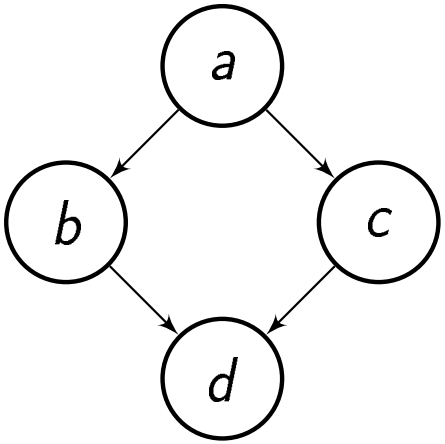
By adding this "learned" clause to our database, we prevent all future cases where this conflict may arise again.

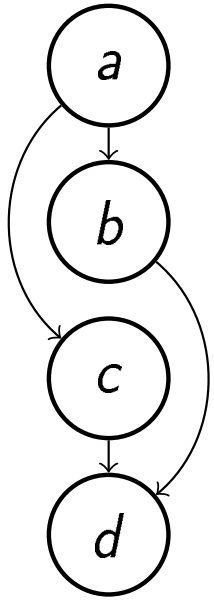
Much, much more optimization

All aspects of SAT algorithms have been studied and analyzed extensively. Choice heuristics and efficient data structure have been developed for decision variable branching, learned clause cut selection, clause management , clause watching, clause reduction, assignment trail recording, random restarting, preprocessing, etc.

What SAT

Here I will present a practical application of SAT solving an NP complete problem.

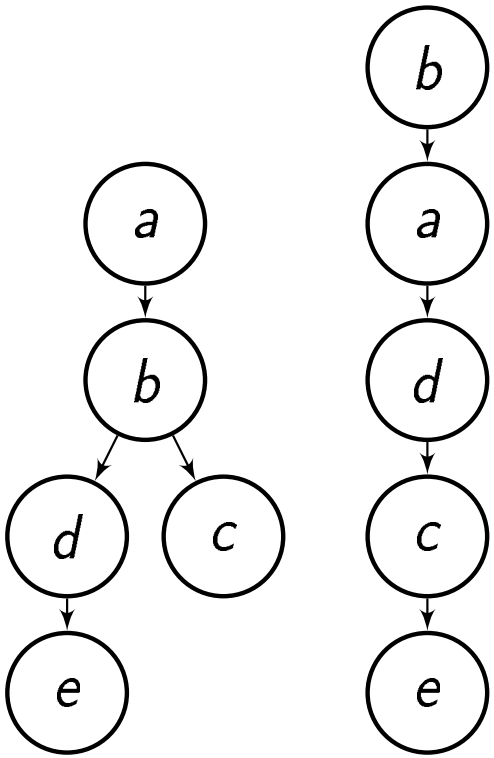


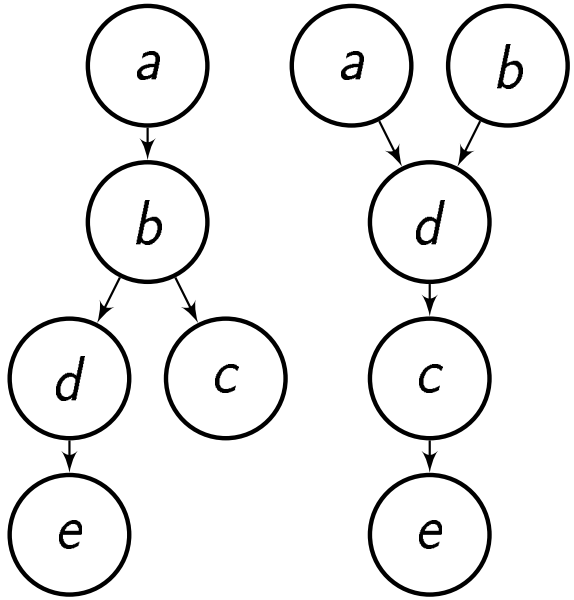


For a DAG, a topological sort is an ordering of its vertices such that if u is before v, then there cannot be an edge from v back to u.
An example of a topological sort of the DAG on the left is abcd, which is depicted on the right.

Poset cover problem

The inverse problem of topological sorting is called the **poset cover problem**, where a set of linear ordering is given, and the goal is to find a minimal set of DAGs that can topological sort out to them. Given the set of linear ordering $\{\texttt{abdce}, \texttt{badce}, \texttt{abcde}, \texttt{abdec}\}$, two minimal poset covers of two DAGs are shown below:





Reduction of poset cover problem to SAT

For a DAG $\mathcal{P}$ and $x \neq y \in \Omega$, we encode with the propositional variable $\mathtt{X}^{\mathcal{P}}_{x,y}$ the case that there is a path from x to y. We must first set some constraints based on the axioms of the DAG properties.

- Two distinct nodes can't have paths both ways: $\bigwedge_{x \neq y \in \Omega} \neg(\mathtt{X}^{\mathcal{P}}_{x,y} \wedge \mathtt{X}^{\mathcal{P}}_{y,x})$
- If x leads to y and y leads to z then it must be that x leads to z: $\bigwedge_{x \neq y \neq z \in \Omega} \mathtt{X}^{\mathcal{P}}_{x,y} \wedge \mathtt{X}^{\mathcal{P}}_{y,z} \rightarrow \mathtt{X}^{\mathcal{P}}_{x,z}$
- Moreover, for a linear DAG, we have on top of that: $\bigwedge_{x \neq y \in \Omega} \mathtt{X}^{\mathcal{P}}_{x,y} \vee \mathtt{X}^{\mathcal{P}}_{y,x}$

If a DAG $\mathcal{P}$ have a topological sort $\mathcal{L}$, we denote that $\mathcal{P} \sqsubseteq \mathcal{L}$, and this property is encoded by the constraints:

$$\mathcal{P} \sqsubseteq \mathcal{L} := \bigwedge_{x \neq y \in \Omega} \mathtt{X}^{\mathcal{P}}_{x,y} \rightarrow \mathtt{X}^{\mathcal{L}}_{x,y}$$

Reduction of poset cover problem to SAT

Poset cover problem

Given a set of linear DAGs Υ, find a set of DAGs C, called a cover, such that $|C|$ is minimal and the union of the topological sorts of DAGs in C is equal to Υ; that is, $\Upsilon = \bigcup_{\mathcal{P} \in C} L(\mathcal{P})$, where L denotes the set of topological sorts of a DAG.

We can reduce the poset cover problem to SAT and set $|C|$ incrementally from 1 to find the minimal cover, but for the equality $\Upsilon = \bigcup_{\mathcal{P} \in C} L(\mathcal{P})$ to hold, we must have by definition that $\Upsilon \subseteq \bigcup_{\mathcal{P} \in C} L(\mathcal{P})$ and $\Upsilon \supseteq \bigcup_{\mathcal{P} \in C} L(\mathcal{P})$.

Reduction of poset cover problem to SAT

A naive reduction is to encode

$$\Upsilon \subseteq \bigcup_{\mathcal{P} \in \mathcal{C}} L(\mathcal{P})$$

as

$$\bigwedge_{\mathcal{L} \in \Upsilon} \bigvee_{\mathcal{P} \in \mathcal{C}} \mathcal{P} \sqsubseteq \mathcal{L}$$

and

$$\Upsilon \supseteq \bigcup_{\mathcal{P} \in \mathcal{C}} L(\mathcal{P})$$

as

$$\bigwedge_{\mathcal{L} \in \overline{\Upsilon}} \bigwedge_{\mathcal{P} \in \mathcal{C}} \mathcal{P} \not\sqsubseteq \mathcal{L}$$

But this latter case causes exponential blowup because $|\overline{\Upsilon}| \in \mathcal{O}(n!)$.

Reduction of poset cover problem to SAT

A better encoding is needed. It also necesarrily exists because poset cover problem was proved to be NP-complete.

Definition

The *adjacent transposition*, or *swap*, relation $\leftrightarrow$ describes the case of "off by one swap" between linear DAGs with shared universe: for linear DAGs $\mathcal{L}_1, \mathcal{L}_2$, we denote this relation by $\mathcal{L}_1 \leftrightarrow \mathcal{L}_2$

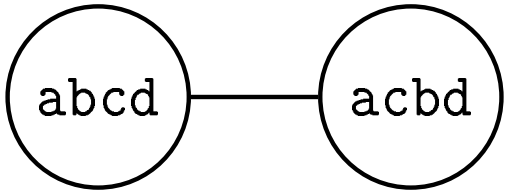
An example of this is $\texttt{abcd} \leftrightarrow \texttt{acbd}$.

Reduction of poset cover problem to SAT

Definition

For a set of linear DAGs Υ, the *swap graph* $G(\Upsilon)$ of it is the undirected graph $(V, E) = (\Upsilon, \leftrightarrow)$.

The swap graph of the previous example $\texttt{abcd} \leftrightarrow \texttt{acbd}$ is:



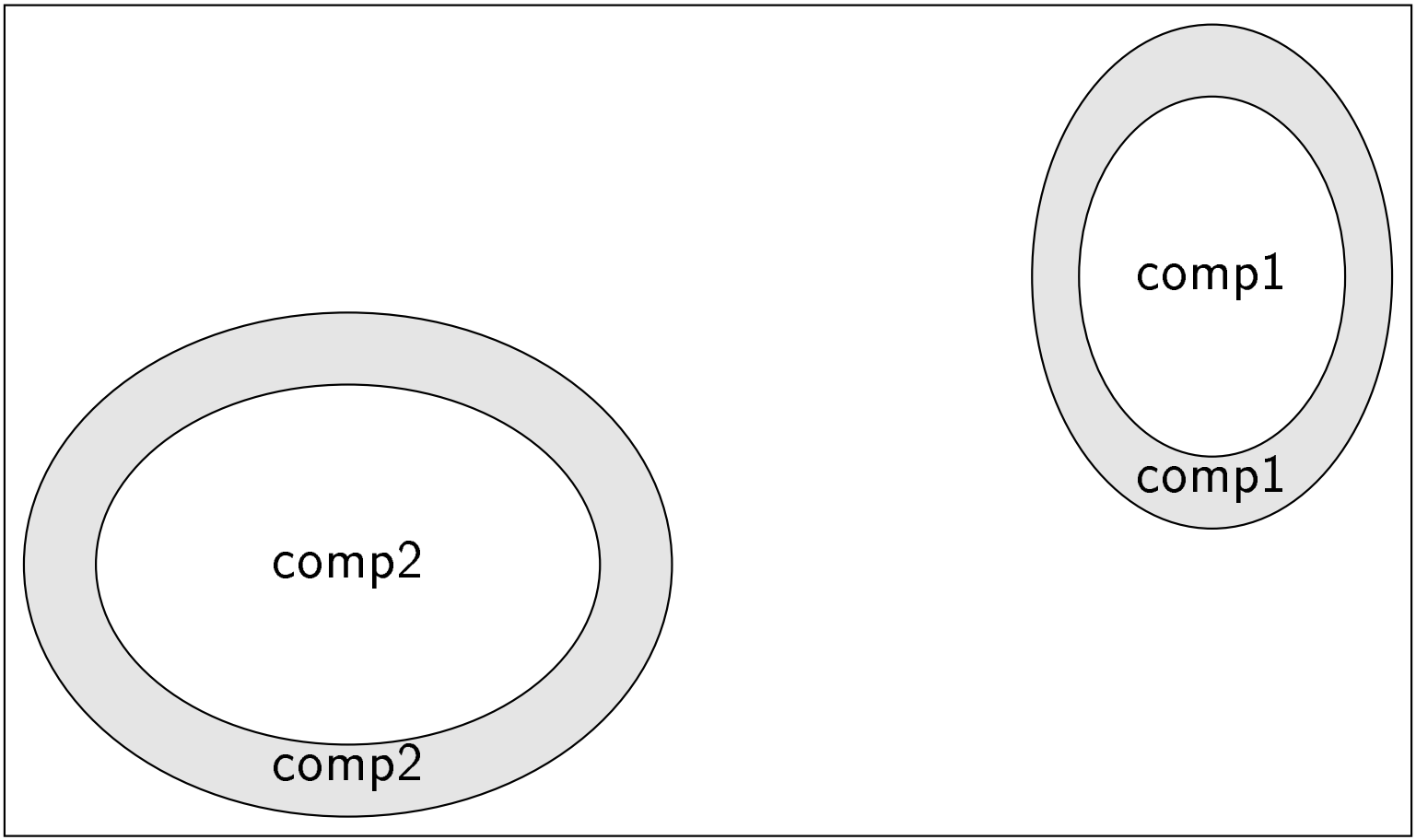
Observation

A swap graph built from the topological sorts of a DAG is connected.

We will exploit this observation for our better reduction.

Reduction of poset cover problem to SAT

We "insulate" each closed connected component $\upsilon \subseteq G(\Upsilon)$ with a "moat" set. We denote by $moat(\Upsilon)$ the union of the moat sets.



Reduction of poset cover problem to SAT

With this information, an alternative way to encode $\Upsilon \supseteq \bigcup_{\mathcal{P} \in \mathcal{C}} L(\mathcal{P})$ is:

$$\bigwedge_{\mathcal{L} \in moat(\Upsilon)} \bigwedge_{\mathcal{P} \in \mathcal{C}} \mathcal{P} \not\sqsubseteq \mathcal{L}$$

And we have found our reduction.

The end

The source code of this slides is available at: `https://github.com/RexYuan/Henwick/blob/master/slides/2023_4_21.tex`

Questions?