Scheduling and SAT

Emmanuel Hebrard

LAAS-CNRS

Toulouse
Outline

1 Introduction

2 Scheduling and SAT Encoding

3 Scheduling and SAT Heuristics

4 Scheduling and SAT Hybrids

5 Conclusion
Outline

1 Introduction
   - Preamble
   - Scheduling Background
   - SAT Background
   - Formulation into SAT

2 Scheduling and SAT Encoding

3 Scheduling and SAT Heuristics

4 Scheduling and SAT Hybrids

5 Conclusion
Scheduling with Boolean Satisfiability
Scheduling with Boolean Satisfiability

- Number of hits for the Google query "Scheduling problem" with ...
Scheduling with Boolean Satisfiability

- Number of hits for the Google query "Scheduling problem" with ...

130,000

"Mixed Integer Programming"  "Constraint Programming"  "Boolean Satisfiability"

OR

"Integer Linear Programming"
Scheduling with Boolean Satisfiability

- Number of hits for the Google query "Scheduling problem" with ...

130,000
"Mixed Integer Programming"
OR
"Integer Linear Programming"

60,000
"Constraint Programming"    "Boolean Satisfiability"
Scheduling with Boolean Satisfiability

- Number of hits for the Google query "Scheduling problem" with …

130,000
"Mixed Integer Programming"
OR
"Integer Linear Programming"

60,000
"Constraint Programming"

21,000
"Boolean Satisfiability"
Scheduling with Boolean Satisfiability

- Important theoretical results
  - [Cook-Levin] theorem: “First” NP-complete problem
  - [Schaefer]’s dichotomy theorem

- Efficient algorithms (CDCL)

- Successful in Circuit design, Model checking, Planning, ...
Scheduling with Boolean Satisfiability

- Important theoretical results
  - [Cook-Levin] theorem: “First” NP-complete problem
  - [Schaefer]’s dichotomy theorem
- Efficient algorithms (CDCL)
- Successful in Circuit design, Model checking, Planning, ...

Association of scheduling and SAT not as natural as MIP or CP
Scheduling with Boolean Satisfiability

- Important theoretical results
  - [Cook-Levin] theorem: “First” NP-complete problem
  - [Schaefer]’s dichotomy theorem
- Efficient algorithms (CDCL)
- Successful in Circuit design, Model checking, Planning, ...

Association of scheduling and SAT not as natural as MIP or CP

- Somewhat counter-intuitive (Boolean vs. Range, logical operator)
Scheduling with Boolean Satisfiability

- Important theoretical results
  - [Cook-Levin] theorem: “First” NP-complete problem
  - [Schaefer]’s dichotomy theorem
- Efficient algorithms (CDCL)
- Successful in Circuit design, Model checking, Planning, ...

Association of scheduling and SAT not as natural as MIP or CP

- Somewhat counter-intuitive (Boolean vs. Range, logical operator)
  - Apparent issue, the numerical aspect can often be avoided
Scheduling with Boolean Satisfiability

- Important theoretical results
  - [Cook-Levin] theorem: “First” NP-complete problem
  - [Schaefer]’s dichotomy theorem
- Efficient algorithms (CDCL)
- Successful in Circuit design, Model checking, Planning, ...

Association of scheduling and SAT not as natural as MIP or CP

- Somewhat counter-intuitive (Boolean vs. Range, logical operator)
  - Apparent issue, the numerical aspect can often be avoided
- Efficiency? SAT Solvers have not always been good

Efficiency? SAT Solvers have not always been good
Scheduling with Boolean Satisfiability

- Important theoretical results
  - [Cook-Levin] theorem: “First” NP-complete problem
  - [Schaefer]’s dichotomy theorem
- Efficient algorithms (CDCL)
- Successful in Circuit design, Model checking, Planning, ...

Association of scheduling and SAT not as natural as MIP or CP

- Somewhat counter-intuitive (Boolean vs. Range, logical operator)
  - Apparent issue, the numerical aspect can often be avoided
- Efficiency? SAT Solvers have not always been good
  - They have made huge progress in the past 10 years
Scheduling with Boolean Satisfiability

- Important theoretical results
  - [Cook-Levin] theorem: “First” NP-complete problem
  - [Schaefer]’s dichotomy theorem
- Efficient algorithms (CDCL)
- Successful in Circuit design, Model checking, Planning, ...

Association of scheduling and SAT not as natural as MIP or CP

- Somewhat counter-intuitive (Boolean vs. Range, logical operator)
  - Apparent issue, the numerical aspect can often be avoided
- Efficiency? SAT Solvers have not always been good
  - They have made huge progress in the past 10 years

- All the approaches discussed here were developed in the last 5 years
Scheduling with Boolean Satisfiability

- Important theoretical results
  - [Cook-Levin] theorem: “First” NP-complete problem
  - [Schaefer]’s dichotomy theorem
- Efficient algorithms (CDCL)
- Successful in Circuit design, Model checking, Planning, ...

Association of scheduling and SAT not as natural as MIP or CP

- Somewhat counter-intuitive (Boolean vs. Range, logical operator)
  - Apparent issue, the numerical aspect can often be avoided
- Efficiency? SAT Solvers have not always been good
  - They have made huge progress in the past 10 years

- All the approaches discussed here were developed in the last 5 years
- Recent progress in SAT algorithms opens new research opportunities
Terminology

- **Tasks** (preemptive, non-preemptive)
- **Resources** (disjunctive, cumulative, reservoir,...)
- **Objectives** (makespan, tardiness, flow time,...)
- **Side constraints** (precedence, time windows, time lags,...)
Scheduling Problems

Terminology
- Tasks (preemptive, non-preemptive)
- Resources (disjunctive, cumulative, reservoir, ...)
- Objectives (makespan, tardiness, flow time, ...)
- Side constraints (precedence, time windows, time lags, ...)

Tip of the iceberg
- SAT-based methods have been applied to a very small subset scheduling problems.
  - Minimization of makespan for non-preemptive tasks and disjunctive resources
Jobshop Scheduling Problem

Problem description

- A set of non-preemptive tasks
Problem description

- A set of non-preemptive tasks
- Organized in jobs (sequences)
Problem description

- A set of non-preemptive tasks
- Organized in jobs (sequences)
- Requiring one of \( m \) disjunctive resources
Jobshop Scheduling Problem

Problem description

- A set of non-preemptive tasks
- Organized in jobs (sequences)
- Requiring one of $m$ disjunctive resources
Jobshop Scheduling Problem

Problem description

- A set of non-preemptive tasks
- Organized in jobs (sequences)
- Requiring one of $m$ disjunctive resources
Problem description

- A set of non-preemptive tasks
- Organized in jobs (sequences)
- Requiring one of $m$ disjunctive resources
Jobshop Scheduling Problem

Problem description

- A set of non-preemptive tasks
- Organized in jobs (sequences)
- Requiring one of $m$ disjunctive resources
- Objective: minimize the total duration ($C_{\text{max}}$)
Boolean Satisfiability (SAT)

Problem
- Boolean variables (atoms)
- Propositional logic formula (often CNF)
Boolean Satisfiability (SAT)

Problem
- Boolean variables (atoms)
- Propositional logic formula (often CNF)
- Literals: $a, \overline{a}$
Boolean Satisfiability (SAT)

Problem

- Boolean variables (atoms)
- Propositional logic formula (often CNF)
- Literals: $a, \overline{a}$
- Clauses: $(\overline{a} \lor \overline{f} \lor g), (\overline{a} \lor \overline{f} \lor g), (\overline{a} \lor b), (b \lor c \lor g)$
# Boolean Satisfiability (SAT)

## Problem

- **Boolean variables** (atoms)
- **Propositional logic formula** (often CNF)
  
  **Literals:** \(a, \overline{a}\)
  
  **Clauses:** \((\overline{a} \lor \overline{f} \lor g), (\overline{a} \lor \overline{f} \lor g), (\overline{a} \lor \overline{b}), (b \lor c \lor g)\)
  
  **Solution:** assignment of atoms satisfying all clauses
Boolean Satisfiability (SAT)

Problem

- Boolean variables (atoms)
- Propositional logic formula (often CNF)
- Literals: $a, \overline{a}$
- Clauses: $(\overline{a} \vee f \vee g), (\overline{a} \vee f \vee g), (\overline{a} \vee b), (b \vee \overline{c} \vee g)$
- Solution: assignment of atoms satisfying all clauses

Algorithms

- Stochastic local search (GSAT, WalkSat,...)
- Survey propagation
- DPLL: Tree search + Unit propagation
- CDCL: Conflict Driven Clause learning
**Boolean Satisfiability (SAT)**

**Problem**
- Boolean variables (atoms)
- Propositional logic formula (often CNF)
- Literals: $a, \overline{a}$
- Clauses: $(\overline{a} \lor \overline{f} \lor g), (\overline{a} \lor \overline{f} \lor g), (\overline{a} \lor b), (b \lor \overline{c} \lor g)$
- Solution: assignment of atoms satisfying all clauses

**Algorithms**
- Stochastic local search (GSAT, WalkSat,...)
# Boolean Satisfiability (SAT)

**Problem**
- Boolean variables (atoms)
- Propositional logic formula (often CNF)
- Literals: $a, \overline{a}$
- Clauses: $(\overline{a} \lor \overline{f} \lor g), \ (\overline{a} \lor \overline{f} \lor g), \ (\overline{a} \lor b), \ (b \lor \overline{c} \lor g)$
- Solution: assignment of atoms satisfying all clauses

**Algorithms**
- Stochastic local search (GSAT, WalkSat,...)
- Survey propagation
Boolean Satisfiability (SAT)

Problem
- Boolean variables (atoms)
- Propositional logic formula (often CNF)
- Literals: $a, \bar{a}$
- Clauses: $(\bar{a} \lor \bar{f} \lor g), (\bar{a} \lor \bar{f} \lor g), (\bar{a} \lor \bar{b}), (b \lor \bar{c} \lor g)$
- Solution: assignment of atoms satisfying all clauses

Algorithms
- Stochastic local search (GSAT, WalkSat,...)
- Survey propagation
- DPLL: Tree search + Unit propagation
Boolean Satisfiability (SAT)

Problem

- Boolean variables (atoms)
- Propositional logic formula (often CNF)
- Literals: $a, \bar{a}$
- Clauses: $(\bar{a} \lor \bar{f} \lor g), (\bar{a} \lor \bar{f} \lor g), (\bar{a} \lor \bar{b}), (b \lor \bar{c} \lor g)$
- Solution: assignment of atoms satisfying all clauses

Algorithms

- Stochastic local search (GSAT, WalkSat,...)
- Survey propagation
- DPLL: Tree search + Unit propagation
- CDCL: Conflict Driven Clause learning
Conflict Driven Clause Leaning (CDCL)

"Evolved" from DPLL

Turning point: clause learning ([GRASP] then [Chaff])

First SAT-Solver competition in 2002

Dive in the "search tree" (make decisions)

Unit propagate: if \( a \) must be true, then \( a \) cannot satisfy a clause

\[ a \lor b \lor c \] effectively becomes \[ b \lor c \]

⋆ continue until a fix point is reached

Until reaching a conflicts (dead-end)

Extract a learned clause

Backjump several levels and unit-propagate the learned clause

Adaptive branching heuristics (weight conflicting literals)

And also: restart, simplify the clause base, forget clauses, etc.
Conflict Driven Clause Leaning (CDCL)

“Evolved” from DPLL

- Turning point: clause learning ([GRASP] then [Chaff])
  - First SAT-Solver competition in 2002
Conflict Driven Clause Leaning (CDCL)

“Evolved” from DPLL

- Turning point: clause learning ([GRASP] then [Chaff])
  - First SAT-Solver competition in 2002

- Dive in the “search tree” (make decisions)
  - Unit propagate: if $a$ must be true, then $\overline{a}$ cannot satisfy a clause
Conflict Driven Clause Leaning (CDCL)

“Evolved” from DPLL
- Turning point: clause learning ([GRASP] then [Chaff])
  - First SAT-Solver competition in 2002

- Dive in the “search tree” (make decisions)
  - Unit propagate: if \( a \) must be true, then \( \overline{a} \) cannot satisfy a clause
  - \( \overline{a} \lor b \lor \overline{c} \) effectively becomes \( b \lor \overline{c} \)
    - continue until a fix point is reached
Conflict Driven Clause Learning (CDCL)

“Evolved” from DPLL

- Turning point: clause learning ([GRASP] then [Chaff])
  - First SAT-Solver competition in 2002

- Dive in the “search tree” (make decisions)
  - Unit propagate: if \( a \) must be true, then \( \overline{a} \) cannot satisfy a clause
  - \( \overline{a} \lor b \lor \overline{c} \) effectively becomes \( b \lor \overline{c} \)
    - continue until a fix point is reached

- Until reaching a conflicts (dead-end)
  - Extract a learned clause
  - Backjump several levels and unit-propagate the learned clause
Conflict Driven Clause Leaning (CDCL)

“Evolved” from DPLL

- Turning point: clause learning ([GRASP] then [Chaff])
  - First SAT-Solver competition in 2002

- Dive in the “search tree” (make decisions)
  - Unit propagate: if \( a \) must be true, then \( \overline{a} \) cannot satisfy a clause
  - \( \overline{a} \lor b \lor \overline{c} \) effectively becomes \( b \lor \overline{c} \)
    - continue until a fix point is reached

- Until reaching a conflicts (dead-end)
  - Extract a learned clause
  - Backjump several levels and unit-propagate the learned clause

- Adaptive branching heuristics (weight conflicting literals)
Conflict Driven Clause Leaning (CDCL)

“Evolved” from DPLL

- Turning point: clause learning ([GRASP] then [Chaff])
  - First SAT-Solver competition in 2002

- Dive in the “search tree” (make decisions)
  - Unit propagate: if \( a \) must be true, then \( \overline{a} \) cannot satisfy a clause
  - \( \overline{a} \lor b \lor \overline{c} \) effectively becomes \( b \lor \overline{c} \)
    - continue until a fix point is reached

- Until reaching a conflicts (dead-end)
  - Extract a learned clause
  - Backjump several levels and unit-propagate the learned clause

- Adaptive branching heuristics (weight conflicting literals)

- And also: restart, simplify the clause base, forget clauses, etc.
**CDCL: Example**

<table>
<thead>
<tr>
<th></th>
<th>$f$</th>
<th>$\overline{a} \lor \overline{f} \lor g$</th>
<th>$c \lor h \lor n \lor \overline{m}$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>$a \lor \overline{b} \lor \overline{h}$</td>
<td>$c \lor l$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$a \lor c$</td>
<td>$d \lor \overline{k} \lor l$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$a \lor \overline{i} \lor \overline{l}$</td>
<td>$d \lor \overline{g} \lor l$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$a \lor \overline{k} \lor \overline{j}$</td>
<td>$\overline{g} \lor n \lor o$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$b \lor d$</td>
<td>$h \lor \overline{o} \lor \overline{j} \lor n$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$b \lor g \lor \overline{n}$</td>
<td>$\overline{i} \lor j$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$b \lor \overline{f} \lor \overline{n} \lor k$</td>
<td>$\overline{d} \lor \overline{l} \lor \overline{m}$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\overline{c} \lor k$</td>
<td>$\overline{e} \lor m \lor \overline{n}$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\overline{c} \lor \overline{k} \lor \overline{i} \lor l$</td>
<td>$\overline{f} \lor h \lor i$</td>
</tr>
</tbody>
</table>
CDCL: Example

\[ \overline{a} \lor \overline{f} \lor g \]
\[ \overline{a} \lor \overline{b} \lor \overline{h} \]
\[ a \lor c \]
\[ a \lor \overline{i} \lor \overline{l} \]
\[ a \lor \overline{k} \lor \overline{j} \]
\[ b \lor d \]
\[ b \lor \overline{g} \lor \overline{n} \]
\[ b \lor \overline{f} \lor \overline{n} \lor k \]
\[ \overline{c} \lor \overline{k} \]
\[ \overline{c} \lor \overline{k} \lor \overline{i} \lor l \]
\[ \overline{c} \lor \overline{k} \lor \overline{i} \lor l \]
\[ \overline{d} \lor \overline{l} \lor \overline{m} \]
\[ \overline{e} \lor m \lor \overline{n} \]
\[ \overline{f} \lor h \lor i \]
CDCL: Example

\[
\begin{array}{c|c|c}
 & f & \\
\hline
a & \rightarrow g & \\
\hline
b & \rightarrow \overline{h} & \\
\hline
\end{array}
\]

\[
\begin{array}{c}
\overline{a} \lor \overline{f} \lor g \\
\overline{a} \lor \overline{b} \lor \overline{h} \\
a \lor c \\
a \lor \overline{i} \lor \overline{l} \\
a \lor \overline{k} \lor \overline{j} \\
b \lor d \\
b \lor g \lor \overline{n} \\
b \lor \overline{f} \lor n \lor k \\
\overline{c} \lor k \\
\overline{c} \lor \overline{k} \lor \overline{i} \lor \overline{l} \\
c \lor h \lor n \lor \overline{m} \\
c \lor \overline{l} \\
d \lor \overline{k} \lor \overline{l} \\
d \lor \overline{g} \lor \overline{l} \\
\overline{g} \lor n \lor \overline{o} \\
h \lor \overline{o} \lor \overline{j} \lor n \\
\overline{i} \lor j \\
\overline{d} \lor \overline{l} \lor \overline{m} \\
\overline{e} \lor m \lor \overline{n} \\
\overline{f} \lor h \lor \overline{i}
\end{array}
\]
CDCL: Example

\[ a \lor \bar{f} \lor g \]
\[ a \lor \bar{b} \lor \bar{h} \]
\[ a \lor c \]
\[ a \lor \bar{i} \lor \bar{l} \]
\[ a \lor \bar{k} \lor \bar{j} \]
\[ b \lor d \]
\[ b \lor g \lor \bar{n} \]
\[ c \lor \bar{k} \]
\[ \bar{c} \lor \bar{k} \lor \bar{\bar{i}} \lor \bar{l} \]
\[ c \lor h \lor n \lor \bar{\bar{m}} \]
\[ c \lor \bar{l} \]
\[ d \lor \bar{k} \lor \bar{l} \]
\[ d \lor \bar{\bar{g}} \lor \bar{l} \]
\[ \bar{g} \lor n \lor \bar{o} \]
\[ h \lor \bar{o} \lor \bar{j} \lor n \]
\[ \bar{i} \lor j \]
\[ \bar{d} \lor \bar{l} \lor \bar{\bar{m}} \]
\[ \bar{e} \lor m \lor \bar{\bar{n}} \]
\[ \bar{\bar{f}} \lor h \lor \bar{i} \]
CDCL: Example

\[ a \lor f \lor g \]
\[ a \lor b \lor h \]
\[ a \lor c \]
\[ a \lor i \lor l \]
\[ a \lor k \lor j \]
\[ b \lor d \]
\[ b \lor g \lor \overline{n} \]
\[ b \lor \overline{f} \lor n \lor k \]
\[ \overline{c} \lor k \]
\[ \overline{c} \lor k \lor \overline{i} \lor l \]
\[ c \lor h \lor n \lor \overline{m} \]
\[ c \lor l \]
\[ d \lor \overline{k} \lor l \]
\[ d \lor \overline{g} \lor l \]
\[ \overline{g} \lor n \lor \overline{o} \]
\[ h \lor \overline{o} \lor \overline{j} \lor n \]
\[ \overline{i} \lor j \]
\[ \overline{d} \lor \overline{l} \lor \overline{m} \]
\[ \overline{e} \lor m \lor \overline{n} \]
\[ \overline{f} \lor h \lor i \]
CDCL: Example

\[
\begin{align*}
\overline{a} \lor \overline{f} \lor g \\
\overline{a} \lor \overline{b} \lor \overline{h} \\
a \lor c \\
a \lor \overline{i} \lor \overline{l} \\
a \lor \overline{k} \lor \overline{j} \\
b \lor d \\
b \lor \overline{g} \lor \overline{n} \\
b \lor \overline{f} \lor n \lor k \\
\overline{c} \lor k \\
\overline{c} \lor \overline{k} \lor \overline{i} \lor \overline{l} \\
c \lor h \lor n \lor \overline{m} \\
c \lor \overline{l} \\
d \lor \overline{k} \lor \overline{l} \\
d \lor \overline{g} \lor \overline{l} \\
\overline{g} \lor n \lor o \\
h \lor \overline{o} \lor \overline{j} \lor n \\
\overline{i} \lor \overline{j} \\
\overline{d} \lor \overline{l} \lor \overline{m} \\
\overline{e} \lor m \lor \overline{n} \\
\overline{f} \lor h \lor i
\end{align*}
\]
CDCL: Example
CDCL: Example

\[
\begin{align*}
\overline{a} \lor \overline{f} \lor g \\
\overline{a} \lor \overline{b} \lor \overline{h} \\
a \lor c \\
a \lor \overline{i} \lor \overline{l} \\
a \lor \overline{k} \lor \overline{j} \\
b \lor d \\
b \lor g \lor \overline{n} \\
b \lor \overline{f} \lor n \lor k \\
\overline{c} \lor k \\
\overline{c} \lor k \lor \overline{i} \lor \overline{l} \\
c \lor h \lor n \lor \overline{m} \\
c \lor \overline{l} \\
d \lor \overline{k} \lor \overline{l} \\
d \lor \overline{g} \lor \overline{l} \\
\overline{g} \lor n \lor \overline{o} \\
h \lor \overline{o} \lor \overline{j} \lor n \\
\overline{i} \lor \overline{j} \\
\overline{d} \lor \overline{l} \lor \overline{m} \\
\overline{e} \lor m \lor \overline{n} \\
\overline{f} \lor h \lor i
\end{align*}
\]
CDCL: Example

\[(h \lor o \lor \bar{j} \lor n)\]
CDCL: Example

\begin{equation}
\overline{h} \lor \overline{o} \lor \overline{j} \lor n
\end{equation}
CDCL: Example

\[(h \lor \neg o \lor \neg j \lor n) \equiv (\neg h \land o \land \neg n) \rightarrow \neg j\]
CDCL: Example

\[ \begin{align*}
  f & \rightarrow g \\
  a & \rightarrow b \\
  b & \rightarrow \overline{h} \rightarrow i \rightarrow j \\
  c & \rightarrow k \rightarrow l \\
  d & \rightarrow \overline{m} \\
  e & \rightarrow \overline{n} \rightarrow o \rightarrow \overline{j} \rightarrow \bot
\end{align*} \]

\[ \begin{align*}
  f, a, b, c, d, e & \\
  g, i, k, l, \overline{m} & \rightarrow \overline{h} \rightarrow j \\
  \overline{n} & \rightarrow o \\
  \overline{j} & \\
  \bot
\end{align*} \]
CDCL: Example

\[ f, a, b, c, d, e \]

\[ g, i, k, l, \overline{m} \]

\[ \overline{n} \rightarrow o \]

\[ \overline{j} \]

\[ \perp \]
CDCL: Example

\[
\begin{align*}
\overline{a} \lor \overline{f} \lor g \\
\overline{a} \lor \overline{b} \lor \overline{h} \\
\overline{a} \lor c \\
\overline{a} \lor \overline{i} \lor \overline{l} \\
\overline{a} \lor \overline{k} \lor \overline{j} \\
b \lor d \\
b \lor g \lor \overline{n} \\
b \lor \overline{f} \lor n \lor k \\
c \lor h \lor n \lor \overline{m} \\
c \lor l \\
d \lor \overline{k} \lor l \\
d \lor \overline{g} \lor l \\
\overline{g} \lor n \lor o \\
h \lor \overline{o} \lor \overline{j} \lor n \\
\overline{i} \lor j \\
\overline{d} \lor \overline{l} \lor \overline{m} \\
\overline{e} \lor m \lor \overline{n} \\
\overline{f} \lor h \lor i
\end{align*}
\]
CDCL: Example

\begin{itemize}
  \item $f$
  \item $a \rightarrow g$
  \item $b \rightarrow \bar{h} \rightarrow i \rightarrow j$
\end{itemize}

\begin{align*}
  \bar{a} \lor \bar{f} \lor g \\
  \bar{a} \lor \bar{b} \lor \bar{h} \\
  a \lor c \\
  a \lor \bar{i} \lor \bar{l} \\
  a \lor \bar{k} \lor \bar{j} \\
  b \lor d \\
  b \lor \bar{g} \lor \bar{n} \\
  b \lor \bar{f} \lor n \lor k \\
  \bar{c} \lor k \\
  \bar{c} \lor \bar{k} \lor \bar{i} \lor \bar{l} \\
  c \lor h \lor n \lor \bar{m} \\
  c \lor \bar{l} \\
  d \lor \bar{k} \lor l \\
  d \lor \bar{g} \lor l \\
  \bar{g} \lor n \lor o \\
  h \lor \bar{o} \lor \bar{j} \lor n \\
  \bar{i} \lor j \\
  \bar{d} \lor \bar{l} \lor \bar{m} \\
  \bar{e} \lor m \lor \bar{n} \\
  \bar{f} \lor h \lor i
\end{align*}
CDCL: Example

\[
\begin{align*}
\overline{a} \lor \overline{f} \lor g \\
\overline{a} \lor \overline{b} \lor \overline{h} \\
a \lor c \\
a \lor \overline{i} \lor \overline{l} \\
a \lor \overline{k} \lor \overline{j} \\
b \lor d \\
b \lor g \lor \overline{n} \\
b \lor \overline{f} \lor \overline{n} \lor k \\
\overline{c} \lor k \\
\overline{c} \lor \overline{k} \lor \overline{i} \lor l \\
c \lor h \lor n \lor \overline{m} \\
c \lor l \\
d \lor \overline{k} \lor l \\
d \lor \overline{g} \lor l \\
\overline{g} \lor n \lor o \\
h \lor \overline{o} \lor \overline{j} \lor n \\
\overline{i} \lor j \\
\overline{d} \lor \overline{l} \lor \overline{m} \\
\overline{e} \lor m \lor \overline{n} \\
\overline{f} \lor h \lor \overline{i} \\
\overline{g} \lor h \lor \overline{j} \lor n
\end{align*}
\]
CDCL: Example

\[
\begin{align*}
\overline{a} \lor f \lor g \\
\overline{a} \lor \overline{b} \lor \overline{h} \\
a \lor c \\
a \lor \overline{i} \lor \overline{l} \\
a \lor \overline{k} \lor \overline{j} \\
b \lor d \\
b \lor g \lor \overline{n} \\
b \lor \overline{f} \lor n \lor k \\
\overline{c} \lor k \\
\overline{c} \lor \overline{k} \lor \overline{i} \lor l \\
c \lor h \lor n \lor \overline{m} \\
c \lor l \\
d \lor \overline{k} \lor l \\
d \lor \overline{g} \lor l \\
\overline{g} \lor n \lor o \\
h \lor \overline{o} \lor \overline{j} \lor n \\
\overline{i} \lor j \\
\overline{d} \lor \overline{l} \lor \overline{m} \\
\overline{e} \lor m \lor \overline{n} \\
\overline{f} \lor h \lor \overline{i} \\
\overline{g} \lor h \lor \overline{j} \lor n
\end{align*}
\]
Adaptive heuristics

- Variable State Independent Decaying Sum (VSIDS)
  - Idea ([Chaff]) weight literals in learned conflicts
  - Decay: favor newer weights
Adaptive heuristics

- **Variable State Independent Decaying Sum (VSIDS)**
  - Idea ([Chaff]) weight literals in learned conflicts
  - Decay: favor newer weights

- **Weighted degree heuristic**
  - On a failure: weight the constraint propagated last
Adaptive heuristics

- Variable State Independent Decaying Sum (VSIDS)
  - Idea ([Chaff]) weight literals in learned conflicts
  - Decay: favor newer weights

- Weighted degree heuristic
  - On a failure: weight the constraint propagated last

- Activity Based Search
  - On a succes: weight the variables whose domain has changed
Outline

1 Introduction

2 Scheduling and SAT Encoding
   - Formulation into SAT
   - Scheduling by encoding into SAT

3 Scheduling and SAT Heuristics

4 Scheduling and SAT Hybrids

5 Conclusion
The way we encode problems into SAT has a huge impact on efficiency

- Encoding of Planning problems
- Encoding of CSP (Direct, Log, AC-encoding)
- Encoding of Pseudo-Boolean (Adder, Sorter)
CNF encoding

- The way we encode problems into SAT has a huge impact on efficiency
  - Encoding of Planning problems
  - Encoding of CSP (Direct, Log, AC-encoding)
  - Encoding of Pseudo-Boolean (Adder, Sorter)
- All encodings are based on CSP formulations
  - Some Boolean variables (e.g., relative orders of tasks)
  - Start time variables (Integer variables)
CNF encoding

- The way we encode problems into SAT has a huge impact on efficiency
  - Encoding of Planning problems
  - Encoding of CSP (Direct, Log, AC-encoding)
  - Encoding of Pseudo-Boolean (Adder, Sorter)

- All encodings are based on CSP formulations
  - Some Boolean variables (e.g., relative orders of tasks)
  - Start time variables (Integer variables)

- Integer variables and precedence constraints
Direct Encoding

Domain

- An atom $i_v$ for each pair $(x_i, v \in D(x_i))$
  
  - $i_v \iff x_i = v$
  
  - $x_i = 1$: 1000
  - $x_i = 2$: 0100
  - $x_i = 3$: 0010
  - $x_i = 4$: 0001

- Must take at least a value: $i_1 \lor i_2 \lor \ldots \lor i_n$

- Must take at most one value: $\bigwedge_{v \neq w \in D(x_i)} \overline{i_v} \lor \overline{i_w}$

Complexity

$O(n^2)$ space: $n(n-1)/2$ binary clauses and one $n$-ary clause.

There are different ways to encode the constraints.
**Direct Encoding**

**Domain**

- An atom $i_v$ for each pair $(x_i, v \in D(x_i))$
  
  $\implies i_v \iff x_i = v$

- $x_i = 1$: 1000
- $x_i = 2$: 0100
- $x_i = 3$: 0010
- $x_i = 4$: 0001

- Must take at least a value: $i_1 \lor i_2 \lor \ldots \lor i_n$
- Must take at most one value: $\bigwedge_{v \neq w \in D(x_i)} i_v \lor \overline{i_w}$

**Complexity**

- $O(n^2)$ space: $n(n-1)/2$ binary clauses and one $n$-ary clause.
- There are different ways to encode the constraints.
Constraints: Tuple Encoding

Example of constraint: \(x_i < x_j\)

<table>
<thead>
<tr>
<th>(x_i)</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>(x_j)</td>
<td>(i_1 \lor \overline{j_1})</td>
<td>(i_2 \lor \overline{j_1})</td>
<td>(i_3 \lor \overline{j_1})</td>
<td>(i_4 \lor \overline{j_1})</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>(i_2 \lor \overline{j_2})</td>
<td></td>
<td>(i_3 \lor \overline{j_2})</td>
<td>(i_4 \lor \overline{j_2})</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>(i_3 \lor \overline{j_3})</td>
<td></td>
<td>(i_4 \lor \overline{j_3})</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td>(i_4 \lor \overline{j_3})</td>
<td></td>
</tr>
</tbody>
</table>
Constraints: Tuple Encoding

Example of constraint: $x_i < x_j$

<table>
<thead>
<tr>
<th>$x_i$</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_j$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>$i_1 \lor j_1$</td>
<td>$i_2 \lor j_1$</td>
<td>$i_3 \lor j_1$</td>
<td>$i_4 \lor j_1$</td>
</tr>
<tr>
<td>2</td>
<td>$i_2 \lor j_2$</td>
<td>$i_3 \lor j_2$</td>
<td>$i_4 \lor j_2$</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>$i_3 \lor j_3$</td>
<td>$i_4 \lor j_3$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>$i_4 \lor j_3$</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Costly (in space) and weak (in propagation)

- $O(n^2)$ binary clauses.
- $i_4(x_i \neq 4)$ and $j_1(x_j \neq 1)$ are inconsistent, but not unit propagated.
### Example of constraint: \( x_i < x_j \)

<table>
<thead>
<tr>
<th>assignment</th>
<th>atom</th>
<th>support</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x_i = 1 )</td>
<td>( \overline{i_1} \lor \overline{j_2} \lor \overline{j_3} \lor \overline{j_4} )</td>
<td></td>
</tr>
<tr>
<td>( x_i = 2 )</td>
<td>( \overline{i_2} \lor \overline{j_3} \lor \overline{j_4} )</td>
<td></td>
</tr>
<tr>
<td>( x_i = 3 )</td>
<td>( \overline{i_3} \lor \overline{j_4} )</td>
<td></td>
</tr>
<tr>
<td>( x_i = 4 )</td>
<td>( \overline{i_4} \lor \bot )</td>
<td></td>
</tr>
</tbody>
</table>
Example of constraint: \( x_i < x_j \)

<table>
<thead>
<tr>
<th>assignment</th>
<th>atom</th>
<th>support</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x_i = 1 )</td>
<td>( \overline{i}_1 \lor \overline{j}_2 \lor \overline{j}_3 \lor \overline{j}_4 )</td>
<td></td>
</tr>
<tr>
<td>( x_i = 2 )</td>
<td>( \overline{i}_2 \lor \overline{j}_3 \lor \overline{j}_4 )</td>
<td></td>
</tr>
<tr>
<td>( x_i = 3 )</td>
<td>( \overline{i}_3 \lor \overline{j}_4 )</td>
<td></td>
</tr>
<tr>
<td>( x_i = 4 )</td>
<td>( \overline{i}_4 \lor \bot )</td>
<td></td>
</tr>
</tbody>
</table>

Same space complexity, better propagation

- \( O(n) \) \( n \)-ary clauses
- \( \overline{i}_4(x_i \neq 4) \) and \( \overline{j}_1(x_j \neq 1) \) are unit clauses.
**Order Encoding [Crawford & Backer 94]**

**Domain**

- An atom $i_v$ for each pair $(x_i, v \in D(x_i))$
  
  ![atom assignments]

- Bound propagation:
  
  - If $x_i \leq v$ then $x_i \leq v + 1$
  
  ![bound propagation formula]
Order Encoding [Crawford & Backer 94]

Domain

- An atom $i_v$ for each pair $(x_i, v \in D(x_i))$
  
  $x_i = 1$: 1111  
  $x_i = 2$: 0111  
  $x_i = 3$: 0011  
  $x_i = 4$: 0001

  $i_v \iff x_i \leq v$

- Bound propagation:
  
  - If $x_i \leq v$ then $x_i \leq v + 1$
  
  $\bigwedge_{v \in D(x_i)} \overline{i_v} \lor i_{v+1}$

Complexity

- $O(n)$ space ($n - 1$ binary clauses)
**Constraints: BC Encoding**

**Example of constraint:** \( x_i < x_j \)

<table>
<thead>
<tr>
<th>relation</th>
<th>clause</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x_i &gt; 0 \Rightarrow x_j &gt; 1 )</td>
<td>( \bot \lor \overline{j_1} )</td>
</tr>
<tr>
<td>( x_i &gt; 1 \Rightarrow x_j &gt; 2 )</td>
<td>( i_1 \lor \overline{j_2} )</td>
</tr>
<tr>
<td>( x_i &gt; 2 \Rightarrow x_j &gt; 3 )</td>
<td>( i_2 \lor \overline{j_3} )</td>
</tr>
<tr>
<td>( x_i &gt; 3 \Rightarrow x_j &gt; 4 )</td>
<td>( i_3 \lor \bot )</td>
</tr>
</tbody>
</table>

Better complexity and same propagation on some linear constraints

\( O(n) \) space (\( n \) binary clauses)

\( i_3, j_1, j_2, j_3 \) are unit clauses.
Example of constraint:  $x_i < x_j$

<table>
<thead>
<tr>
<th>relation</th>
<th>clause</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_i &gt; 0 \Rightarrow x_j &gt; 1$</td>
<td>$\bot \lor \overline{j_1}$</td>
</tr>
<tr>
<td>$x_i &gt; 1 \Rightarrow x_j &gt; 2$</td>
<td>$i_1 \lor \overline{j_2}$</td>
</tr>
<tr>
<td>$x_i &gt; 2 \Rightarrow x_j &gt; 3$</td>
<td>$i_2 \lor \overline{j_3}$</td>
</tr>
<tr>
<td>$x_i &gt; 3 \Rightarrow x_j &gt; 4$</td>
<td>$i_3 \lor \bot$</td>
</tr>
</tbody>
</table>

Better complexity and same propagation on some linear constraints

- $O(n)$ space ($n$ binary clauses)
- $i_3(x_i \leq 3)$ and $\overline{j_1}(x_j > 1)$ are unit clauses.
Domain

- An atom $i_k$ for each value in $[1, \ldots, \lfloor \log_2 \text{ub} \rfloor]$ (assuming $D(x) = [0, \ldots, \text{ub}]$)

$$\sum_{k=1}^{\text{ub}} 2^k * i_k = v \iff x_i = v$$

- For interval domains, no need for extra clauses

\[
x_i = 0: \quad 00 \\
x_i = 1: \quad 01 \\
x_i = 2: \quad 10 \\
x_i = 3: \quad 11
\]
Log Encoding [Walsh 00]

Domain

- An atom $i_k$ for each value in $[1, \ldots, \lfloor \log_2 ub \rfloor]$ (assuming $D(x) = [0, \ldots, ub]$)

$$\sum_{k=1}^{ub} 2^k * i_k = v \iff x_i = v$$

- For interval domains, no need for extra clauses

Complexity

- $O(\log_2 n)$ space

Propagation

- Encoding constraints is trickier, and less powerful
Other Encodings

- Mix of direct and order encoding [lazy-FD, Numberjack]
- Mix of AC and log encoding [Gavanelli 2007]
- Mix of order and log encoding [Sugar, Tamura et al. 2006]
Other Encodings

Many more!

- Mix of direct and order encoding [lazy-FD, Numberjack]
- Mix of AC and log encoding [Gavanelli 2007]
- Mix of order and log encoding [Sugar, Tamura et al. 2006]
  - Log encoding in a base $B$ and order encoding inside a digit
  - Excellent results on scheduling benchmarks! (with CDCL solvers)
Order Encoding, Now and Then

Progress of SAT solvers

- From a few hundreds variables in the 90’s to millions now
Progress of SAT solvers

- From a few hundreds variables in the 90’s to millions now

[Crawford & Backer 94]

- Instances from Sadeh, with 10 jobs, 5 operations each (45m cutoff)
- Tableau solved 90% of the instances (about 2 min when it did)
## Order Encoding, Now and Then

### Progress of SAT solvers
- From a few hundreds variables in the 90’s to millions now

**[Crawford & Backer 94]**
- Instances from Sadeh, with 10 jobs, 5 operations each (45m cutoff)
- Tableau solved 90% of the instances (about 2 min when it did)

**[Tamura, Tanjo & Banbara]**
- Same instances used during the CSP Solver Competition
- Similar model, hardware of course incomparable, MiniSat
Order Encoding, Now and Then

Progress of SAT solvers
- From a few hundreds variables in the 90’s to millions now

[Crawford & Backer 94]
- Instances from Sadeh, with 10 jobs, 5 operations each (45m cutoff)
- Tableau solved 90% of the instances (about 2 min when it did)

[Tamura, Tanjo & Banbara]
- Same instances used during the CSP Solver Competition
- Similar model, hardware of course incomparable, MiniSat
- The hardest instance requires a few 100s conflicts at the most
Closing the Open Shop

Instances

- [Gueret & Prins]: hard for local search, extremely easy for SAT/CP
- [Taillard]: Large, but relatively easy
- [Brucker]: Three open instances

Results

All instances solved and proved optimal. The two hardest instances were decomposed into 120 subproblems, and required up to 13h to solve. First approach to close the open shop!
Instances

- [Gueret & Prins]: hard for local search, extremely easy for SAT/CP
- [Taillard]: Large, but relatively easy
- [Brucker]: Three open instances

Results

- All instances solved and proved optimal
  - The two hardest instances were decomposed into 120 subproblems, and required up to 13h to solve
Closing the Open Shop

Instances
- [Gueret & Prins]: hard for local search, extremely easy for SAT/CP
- [Taillard]: Large, but relatively easy
- [Brucker]: Three open instances

results
- All instances solved and proved optimal
  - The two hardest instances were decomposed into 120 subproblems, and required up to 13h to solve
- First approach to close the open shop!
Solving vs. Encoding

[Tamura et al.]'s encoding is better than order encoding.

However, the huge difference with respect to [Crawford & Backer 94] is due to the solver.

It is now possible to efficiently solve some scheduling problem simply by formulating it as a CNF formula.
Solving vs. Encoding

- Tamura et al.’s encoding is better than order encoding
Solving vs. Encoding

- [Tamura et al.]'s encoding is better than order encoding
  - However, the huge difference with respect to [Crawford & Backer 94] is due to the solver
Solving vs. Encoding

- Tamura et al.'s encoding is better than order encoding
  - However, the huge difference with respect to Crawford & Backer 94 is due to the solver

- It is now possible to efficiently solve some scheduling problem simply by formulating it as a CNF formula
A SAT-like Approach [Grimes & Hebrard 09]

- CSP Solver Competition: scheduling benchmarks
A SAT-like Approach [Grimes & Hebrard 09]

- CSP Solver Competition: scheduling benchmarks
  - Some hard instances
  - Generic format (XCSP), the notions of resource is lost, no global constraint

Yet many solvers solved them ([Sugar], [Choco], [Mistral])

Experiment with Weighted degree [Boussemart et al. 04]

Similar simple model in [Mistral], same observation [Grimes]

Open shop instances closed by [Tamura et al.] can be solved to optimality in a few minutes

Are adaptive heuristics all that we need to solve disjunctive scheduling problems?
CSP Solver Competition: scheduling benchmarks
- Some hard instances
- Generic format (XCSP), the notions of resource is lost, no global constraint
- Yet many solvers solved them ([Sugar], [Choco], [Mistral])
A SAT-like Approach [Grimes & Hebrard 09]

- CSP Solver Competition: scheduling benchmarks
  - Some hard instances
  - Generic format (XCSP), the notions of resource is lost, no global constraint
  - Yet many solvers solved them ([Sugar], [Choco], [Mistral])
- Experiment with Weighted degree [Boussemart et al. 04]
A SAT-like Approach [Grimes & Hebrard 09]

- CSP Solver Competition: scheduling benchmarks
  - Some hard instances
  - Generic format (XCSP), the notions of resource is lost, no global constraint
  - Yet many solvers solved them ([Sugar], [Choco], [Mistral])

- Experiment with Weighted degree [Boussemart et al. 04]
  - Similar simple model in [Mistral], same observation [Grimes]
A SAT-like Approach [Grimes & Hebrard 09]

- CSP Solver Competition: scheduling benchmarks
  - Some hard instances
  - Generic format (XCSP), the notions of resource is lost, no global constraint
  - Yet many solvers solved them ([Sugar], [Choco], [Mistral])
- Experiment with Weighted degree [Boussemart et al. 04]
  - Similar simple model in [Mistral], same observation [Grimes]
  - Open shop instances closed by [Tamura et al.] can be solved to optimality in a few minutes
A SAT-like Approach [Grimes & Hebrard 09]

- CSP Solver Competition: scheduling benchmarks
  - Some hard instances
  - Generic format (XCSP), the notions of resource is lost, no global constraint
  - Yet many solvers solved them ([Sugar], [Choco], [Mistral])

- Experiment with Weighted degree [Boussemart et al. 04]
  - Similar simple model in [Mistral], same observation [Grimes]
  - Open shop instances closed by [Tamura et al.] can be solved to optimality in a few minutes

- Are adaptive heuristics all that we need to solve disjunctive scheduling problems?
A Variable for the start time of each task: \( t_i \in [0, \ldots, C_{\text{max}}] \).

- **Precedence constraints:** \( t_i + p_i \leq t_{i+1} \).
Constraint Model

- A Variable for the start time of each task: \( t_i \in [0, \ldots, C_{\text{max}}] \).
  - Precedence constraints: \( t_i + p_i \leq t_{i+1} \).

- A Boolean Variable standing for the relative order of each pair of conflicting tasks (disjunct):
  - Binary Disjunctive constraints: 
    \[
    b_{ij} = \begin{cases} 
    0 \iff t_i + p_i \leq t_j \\
    1 \iff t_j + p_j \leq t_i 
    \end{cases}
    \]
Search Strategy

Adaptive heuristic
Branch on Boolean variables only (order tasks on machines)
Minimum domain over weighted degree [Boussemart et al. 04]

Guided search
Follow the branch corresponding to the best solution [Beck 07]
≃ phase-saving heuristic in SAT [Pipatsrisawat & Darwiche 07]

Restarts
Geometric [Walsh 99], nogoods on restarts [Lecoutre et al. 07]
Almost no problem specific method
Search Strategy

Adaptive heuristic

- Branch on Boolean variables only (order tasks on machines)
- Minimum domain over weighted degree [Boussemart et al. 04]
## Search Strategy

### Adaptive heuristic
- Branch on Boolean variables only (order tasks on machines)
- Minimum domain over weighted degree [Boussemart et al. 04]

### Guided search
- Follow the branch corresponding to the best solution [Beck 07]
- $\simeq$ phase-saving heuristic in SAT [Pipatsrisawat & Darwiche 07]
Search Strategy

### Adaptive heuristic
- Branch on Boolean variables only (order tasks on machines)
- Minimum domain over weighted degree [Boussemart et al. 04]

### Guided search
- Follow the branch corresponding to the best solution [Beck 07]
- \(\simeq\) phase-saving heuristic in SAT [Pipatsrisawat & Darwiche 07]

### Restarts
- Geometric [Walsh 99], nogoods on restarts [Lecoutre et al. 07]
## Search Strategy

### Adaptive heuristic
- Branch on Boolean variables only (order tasks on machines)
- Minimum domain over weighted degree [Boussemart et al. 04]

### Guided search
- Follow the branch corresponding to the best solution [Beck 07]
- \(\simeq\) phase-saving heuristic in SAT [Pipatsrisawat & Darwiche 07]

### Restarts
- Geometric [Walsh 99], nogoods on restarts [Lecoutre et al. 07]
- Almost no problem specific method
CP or SAT?

Many similarities with SAT:
- Search variables are Boolean
- Propagation is very basic
- SAT-based search strategies
CP or SAT?

Many similarities with SAT:
- Search variables are Boolean
- Propagation is very basic
- SAT-based search strategies

Some differences
- Faster propagation, but no clause learning
CP or SAT?

- Many similarities with SAT:
  - Search variables are Boolean
  - Propagation is very basic
  - SAT-based search strategies

Some differences

- Faster propagation, but no clause learning
- Restarts + weighted degree “simulates” CDCL behavior?
Experiment on Jobshop and Variants
Experiment on Jobshop and Variants

- Sequence-dependent setup times
  - Transition between tasks on a machine
Experiment on Jobshop and Variants

- Sequence-dependent setup times
  - Transition between tasks on a machine
  - Add the transition times in the disjunct
Experiment on Jobshop and Variants

- **Sequence-dependent setup times**
  - Transition between tasks on a machine
  - Add the transition times in the disjunct

- **Maximum time lags**
  - Maximum duration between consecutive tasks in a job
Experiment on Jobshop and Variants

- **Sequence-dependent setup times**
  - Transition between tasks on a machine
  - Add the transition times in the disjunct

- **Maximum time lags**
  - Maximum duration between consecutive tasks in a job
  - Precedences with negative durations
Experiment on Jobshop and Variants

- Sequence-dependent setup times
  - Transition between tasks on a machine
  - Add the transition times in the disjunct
- Maximum time lags
  - Maximum duration between consecutive tasks in a job
  - Precedences with negative durations
- Just in Time scheduling
  - Penalties for earliness and tardiness of each job
Experiment on Jobshop and Variants

- Sequence-dependent setup times
  - Transition between tasks on a machine
  - Add the transition times in the disjunct

- Maximum time lags
  - Maximum duration between consecutive tasks in a job
  - Precedences with negative durations

- Just in Time scheduling
  - Penalties for earliness and tardiness of each job
  - Simple decomposition to express the new objective
Experimental Protocol

- This simple model was run on several standard benchmarks
  - 1 hour cutoff
  - 10 random runs, we take the best
- Best known results on each benchmark (LS, CP, MIP)
  - The cutoff may be different
  - The hardware is different

\[
\text{Average } \% \text{ deviation (with respect to a method } M \text{ in } \{\text{MIP, CP, LS}\}) = \\
100 \times \sum_{\text{instance}} \text{M objective}(x) - \text{SAT objective}(x) \times \# \text{instances} \times \text{best objective}(x)
\]

- Negative: how much worse than M (when it is)
- Positive: how much better than M (when it is)
Experimental Protocol

- This simple model was run on several standard benchmarks
  - 1 hour cutoff
  - 10 random runs, we take the best
- Best known results on each benchmark (LS, CP, MIP)
  - The cutoff may be different
  - The hardware is different
- Average % deviation (with respect to a method M in \{MIP, CP, LS\})
  - \[100 \times \sum_{\text{instance } x} \frac{M \text{ objective}(x) - SAT \text{ objective}(x)}{\# \text{instances} \times \text{best objective}(x)}\]
Experimental Protocol

- This simple model was run on several standard benchmarks
  - 1 hour cutoff
  - 10 random runs, we take the best

- Best known results on each benchmark (LS, CP, MIP)
  - The cutoff may be different
  - The hardware is different

- Average % deviation (with respect to a method $M$ in \{MIP, CP, LS\})
  - $100 \times \sum_{\text{instance } x} \frac{M \text{ objective}(x) - SAT \text{ objective}(x)}{\#\text{instances} \times \text{best objective}(x)}$
    - Negative: how much worse than $M$ (when it is)
Experimental Protocol

- This simple model was run on several standard benchmarks
  - 1 hour cutoff
  - 10 random runs, we take the best
- Best known results on each benchmark (LS, CP, MIP)
  - The cutoff may be different
  - The hardware is different
- Average % deviation (with respect to a method $M$ in \{MIP, CP, LS\})
  - \[
  100 \times \sum_{\text{instance } x} \frac{M \text{ objective}(x) - SAT \text{ objective}(x)}{\#\text{instances} \times \text{best objective}(x)}
  \]
  - Negative: how much worse than $M$ (when it is)
  - Positive: how much better than $M$ (when it is)
Jobshop - $C_{max}$ - Taillard

- LS [1]
- CP [2]
- MIP (no data)

[1] i-TSAB (Tabu Search)
E. Nowicki and C. Smutnicki
J. of Scheduling (2005)

[2] SGMPCS (Ilog Scheduler)
J. C. Beck
JAIR (2007)
Jobshop with setup times - $C_{max}$ - Brucker & Thiele

![Graph showing % Deviation for JSP, Setup, Time-lag, No-wait, and JiT for different algorithms: LS [3], CP [2], SGMPCS (Ilog Scheduler) [2], and MIP (no data).]

- LS [3]: GA / Tabu Search
  Gonzalez, Vela, and Varela
  ICAPS (2008)

- CP [2]: SGMPCS (Ilog Scheduler)
  J. C. Beck
  JAIR (2007)
Jobshop with time lags - $C_{max}$ - Lawrence (modified)

% Deviation

- JSP
- Setup
- Time-lag
- No-wait
- JiT

- LS [4]
- CP [2]
- MIP (no data)

Caumond, Lacomme
and Tchernev
Computers & OR (2008)

[2] SGMPCS (Ilog Scheduler)
J. C. Beck
JAIR (2007)
"No-wait" Jobshop - $C_{max}$ - Lawrence

- LS [5]: Hybrid Tabu Search
  Bozejko and Makuchowski
  Computers & IE (2009)

- CP [2]: SGMPCS (Ilog Scheduler)
  J. C. Beck
  JAIR (2007)

- MIP [6]: CPLEX
  J. J. J. van den Broek
  PhD thesis (2009)
Jobshop - earliness/tardiness - Beck & Refalo; Morton & Pentico

% Deviation

JSP Setup Time-lag No-wait JiT


LNS (CP/LP)
Danna and Peron

Hybrid CP/LP
Beck and Refalo

Cplex
Danna, Rothberg, Le Pape
CPAIOR (2003)
SAT Strategies

- Often comparable or better than the state of the art
SAT Strategies

- Often comparable or better than the state of the art
  - On benchmarks that are more favorable?
SAT Strategies

- Often comparable or better than the state of the art
  - On benchmarks that are more favorable?
  - On benchmarks that received less attention?
SAT Strategies

- Often comparable or better than the state of the art
  - On benchmarks that are more favorable?
  - On benchmarks that received less attention?
- Adaptive heuristics are extremely powerful
SAT Strategies

- Often comparable or better than the state of the art
  - On benchmarks that are more favorable?
  - On benchmarks that received less attention?
- Adaptive heuristics are extremely powerful
  - Effective at detecting bottlenecks
SAT Strategies

- Often comparable or better than the state of the art
  - On benchmarks that are more favorable?
  - On benchmarks that received less attention?

- Adaptive heuristics are extremely powerful
  - Effective at detecting bottlenecks
  - Often better than dedicated CP approaches to \textit{prove} optimality
SAT Strategies

- Often comparable or better than the state of the art
  - On benchmarks that are more favorable?
  - On benchmarks that received less attention?

- Adaptive heuristics are extremely powerful
  - Effective at detecting bottlenecks
  - Often better than dedicated CP approaches to prove optimality
    - Even this “pseudo” learning helps!
Outline

1 Introduction

2 Scheduling and SAT Encoding

3 Scheduling and SAT Heuristics

4 Scheduling and SAT Hybrids
   - Lazy clause generation
   - Satisfiability Modulo Theories

5 Conclusion
Pure reformulation is surprisingly efficient
SAT Hybrids

- Pure reformulation is surprisingly efficient
- However, simply using an adaptive heuristic and restart seems at least as good
SAT Hybrids

- Pure reformulation is surprisingly efficient
- However, simply using an adaptive heuristic and restart seems at least as good

Hybridization
SAT Hybrids

- Pure reformulation is surprisingly efficient
- However, simply using an adaptive heuristic and restart seems at least as good

Hybridization

- SAT-based learning AND CP-based propagation
Pure reformulation is surprisingly efficient
However, simply using an adaptive heuristic and restart seems at least as good

Hybridization

- SAT-based learning AND CP-based propagation
  - What is the best tradeoff?
Pure reformulation is surprisingly efficient
However, simply using an adaptive heuristic and restart seems at least as good

Hybridization

- SAT-based learning AND CP-based propagation
  - What is the best tradeoff?
  - Does there need to be a tradeoff?
SAT Hybrids

- Pure reformulation is surprisingly efficient
- However, simply using an adaptive heuristic and restart seems at least as good

Hybridization

- SAT-based learning AND CP-based propagation
  - What is the best tradeoff?
  - Does there need to be a tradeoff?

- Lazy Clause Generation
- SAT Modulo Theories
Lazy Clause Generation [Ohrimenko, Stuckey & Codish 07] - [Feydy & Stuckey 09]

Architecture

- Channel a CP and SAT representations
  - Search and propagation in CP
  - Efficient domain representation and propagators
Lazy Clause Generation [Ohrimenko, Stuckey & Codish 07] - [Feydy & Stuckey 09]

**Architecture**

- Channel a CP and SAT representations
  - Search and propagation in CP
  - Efficient domain representation and propagators
    - Produce clauses to explain the pruning
    - Just enough to extract a conflict
Lazy Clause Generation [Ohrimenko, Stuckey & Codish 07] - [Feydy & Stuckey 09]

Architecture

- Channel a CP and SAT representations
  - Search and propagation in CP
  - Efficient domain representation and propagators
    - Produce clauses to explain the pruning
    - Just enough to extract a conflict
  - The SAT formulation is generated lazily (learned during search)
Lazy Clause Generation

Architecture

- Search
- Domains
- Propagators
- Unit Literals
- Clause Database
Lazy Clause Generation

Architecture

- Search
  - decision
- Domains
- Propagators
- Unit Literals
- Clause Database
Lazy Clause Generation

Architecture

Search

Domains

Propagators

decision

filtering

Unit Literals

Clause Database
Lazy Clause Generation

Architecture

- **Search**
  - decision

- **Domains**
  - filtering

- **Propagators**

- **Clause Database**
  - Unit Literals

Explanation (clause generation)
Lazy Clause Generation

Architecture

- **Search**
  - decision
- **Domains**
  - filtering
- **Propagators**
  - explanation (clause generation)
- **Clause Database**
  - unit literals
  - unit propagation
Lazy Clause Generation

Architecture

- Search
  - decision
  - filtering
- Domains
- Propagators
  - explanation (clause generation)
- Unit Literals
  - unit propagation
- Clause Database

Channeling

Diagram showing the integration of various components in the lazy clause generation architecture.
### Lazy-FD: Example

$x_i < x_j$

- **Initial representation**

<table>
<thead>
<tr>
<th></th>
<th>CP view</th>
<th>SAT view</th>
</tr>
</thead>
<tbody>
<tr>
<td>$D(x_i)$</td>
<td>${1, \ldots, 4}$</td>
<td>$i_1 \lor i_2, i_2 \lor i_3$</td>
</tr>
<tr>
<td>$D(x_j)$</td>
<td>${2, \ldots, 5}$</td>
<td>$j_2 \lor j_3, j_3 \lor j_4$</td>
</tr>
<tr>
<td>constraint</td>
<td>$x_i &lt; x_j$</td>
<td></td>
</tr>
<tr>
<td>constraint</td>
<td>$C(x_i, x_k, \ldots)$</td>
<td></td>
</tr>
</tbody>
</table>
Some constraint reduces the domain of $x_i$ to $\{2, \ldots, 5\}$

<table>
<thead>
<tr>
<th>Constraint</th>
<th>CP View</th>
<th>SAT View</th>
</tr>
</thead>
<tbody>
<tr>
<td>$D(x_i)$</td>
<td>${2, \ldots, 4}$</td>
<td>$i_1 \lor i_2, i_2 \lor i_3$</td>
</tr>
<tr>
<td>$D(x_j)$</td>
<td>${2, \ldots, 5}$</td>
<td>$j_2 \lor j_3, j_3 \lor j_4$</td>
</tr>
<tr>
<td>$x_i &lt; x_j$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$C(x_i, x_k, \ldots)$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Lazy-FD: Example

$x_i < x_j$

- An explanation clause $T \lor \overline{i_1}$ is produced, and the unit literal $\overline{i_1}$ is propagated

<table>
<thead>
<tr>
<th></th>
<th>CP view</th>
<th>SAT view</th>
</tr>
</thead>
<tbody>
<tr>
<td>$D(x_i)$</td>
<td>${2, \ldots, 4}$</td>
<td>$\overline{i_1} \lor i_2, \overline{i_2} \lor i_3$</td>
</tr>
<tr>
<td>$D(x_j)$</td>
<td>${2, \ldots, 5}$</td>
<td>$\overline{j_2} \lor \overline{j_3}, j_3 \lor j_4$</td>
</tr>
<tr>
<td>constraint</td>
<td>$x_i &lt; x_j$</td>
<td>$T \lor \overline{i_1}$</td>
</tr>
<tr>
<td>constraint</td>
<td>$C(x_i, x_k, \ldots)$</td>
<td></td>
</tr>
</tbody>
</table>
**Lazy-FD: Example**

$x_i < x_j$

- The propagator for $x_i < x_j$ is triggered and reduces the domain of $x_j$

<table>
<thead>
<tr>
<th></th>
<th>CP view</th>
<th>SAT view</th>
</tr>
</thead>
<tbody>
<tr>
<td>$D(x_i)$</td>
<td>${2, \ldots, 4}$</td>
<td>$i_1 \lor i_2$, $i_2 \lor i_3$</td>
</tr>
<tr>
<td>$D(x_j)$</td>
<td>${3, \ldots, 5}$</td>
<td>$j_2 \lor j_3$, $j_3 \lor j_4$</td>
</tr>
<tr>
<td>constraint</td>
<td>$x_i &lt; x_j$</td>
<td></td>
</tr>
<tr>
<td>constraint</td>
<td>$C(x_i, x_k, \ldots)$</td>
<td>$T \lor \overline{i_1}$</td>
</tr>
</tbody>
</table>
$x_i < x_j$

- An explanation clause is also produced

<table>
<thead>
<tr>
<th></th>
<th>CP view</th>
<th>SAT view</th>
</tr>
</thead>
<tbody>
<tr>
<td>$D(x_i)$</td>
<td>${2, \ldots, 4}$</td>
<td>$\overline{i_1} \lor i_2, \overline{i_2} \lor i_3$</td>
</tr>
<tr>
<td>$D(x_j)$</td>
<td>${3, \ldots, 5}$</td>
<td>$\overline{j_2} \lor \overline{j_3}, \overline{j_3} \lor j_4$</td>
</tr>
<tr>
<td>constraint</td>
<td>$x_i &lt; x_j$</td>
<td>$i_i \lor \overline{j_2}$</td>
</tr>
<tr>
<td>constraint</td>
<td>$C(x_i, x_k, \ldots)$</td>
<td>$T \lor \overline{i_1}$</td>
</tr>
</tbody>
</table>
Results on Resource Constrained Project Scheduling Problem (RCPSP) [Schutt, Feydy, Stuckey & Wallace 09]

Resource Constrained Project Scheduling Problem (RCPSP)
- Cumulative resources, each task has a demand $r_k$ for the resource $k$
Resource Constrained Project Scheduling Problem (RCPSP) [Schutt, Feydy, Stuckey & Wallace 09]

Resource Constrained Project Scheduling Problem (RCPSP)

- Cumulative resources, each task has a demand $r_k$ for the resource $k$

Model

- Formulated using sums on the order encoding
- A fixed number of runs with a dedicated heuristic, then VSIDS (adaptive heuristic)
Results on Resource Constrained Project Scheduling Problem (RCPSP) [Schutt, Feydy, Stuckey & Wallace 09]

<table>
<thead>
<tr>
<th>Resource Constrained Project Scheduling Problem (RCPSP)</th>
</tr>
</thead>
<tbody>
<tr>
<td>● Cumulative resources, each task has a demand $r_k$ for the resource $k$</td>
</tr>
</tbody>
</table>

**Model**

| ● Formulated using sums on the order encoding |
| ● A fixed number of runs with a dedicated heuristic, then VSIDS (adaptive heuristic) |

**Results**

| ● Favorable comparison with state of the art approaches |
| ▶ MCS (implemented on top of Ilog-Scheduler [Laborie 05]) |
| ▶ CP approach by [Liess & Michelon 08] |
| ▶ MIP approach by [Koné et al.] |
Results on Resource Constrained Project Scheduling Problem (RCPSP) [Schutt, Feydy, Stuckey & Wallace 09]

Resource Constrained Project Scheduling Problem (RCPSP)
- Cumulative resources, each task has a demand $r_k$ for the resource $k$

Model
- Formulated using sums on the order encoding
- A fixed number of runs with a dedicated heuristic, then VSIDS (adaptive heuristic)

Results
- Favorable comparison with state of the art approaches
  - MCS (implemented on top of Ilog-Scheduler [Laborie 05])
  - CP approach by [Liess & Michelon 08]
  - MIP approach by [Koné et al.]
- 54 open instances closed!
SAT Modulo Theories (SMT)

- Framework to hybridize dedicated solvers (Theories, or **T-Solvers**) with CDCL solvers
  - T-Solver view: a set of *propositions* each represented by a literal in $F$
  - CDCL-Solver view: a CNF formula $F$ partially representing the problem
- CDCL-Solver makes decisions and analyzes the conflicts
- T-Solver detects conflicts and/or propagates and generates explanation clauses
SMT Solver

Architecture

- Search
- Literals
- Clauses Database
- T-Solver
SMT Solver

Architecture

Search

\[ \text{decision} \]

Literals

Clauses Database

T-Solver
Architecture

Search

decision

Literals

Unit propagation

Clauses Database

T-Solver

SMT Solver
SMT Solver

Architecture

- Search
- Literals
- Clauses Database
- T-Solver

Relations:
- Decision: Search to Literals
- Channeling: Literals to T-Solver
- Propagation: Clauses Database to Literals
- Unit Propagation: Clauses Database to Literals
Several Theories

T-Solvers

- Linear Real Arithmetic,
- Arrays,
- Bit-Vectors,
- Equality with Uninterpreted Functions,
Several Theories

**T-Solvers**

- Linear Real Arithmetic,
- Arrays,
- Bit-Vectors,
- Equality with Uninterpreted Functions,
- **Difference Logic** (i.e. formulas contain atoms of the form $x - y \leq k$).
Several Theories

T-Solvers
- Linear Real Arithmetic,
- Arrays,
- Bit-Vectors,
- Equality with Uninterpreted Functions,
- **Difference Logic** (i.e. formulas contain atoms of the form $x - y \leq k$).

SMT for scheduling
- Satisfiability Modulo **Difference Logic**.
Example: Jobshop Scheduling

problem

\[
\begin{align*}
   s_1 - s_2 &\leq -2 \\
   s_3 - s_4 &\leq -4 \\
   s_5 - s_6 &\leq -5 \\
   s_2 - z &\leq -5 \\
   s_4 - z &\leq -4 \\
   s_6 - z &\leq -3 \\
   a - s_1 &\leq 0 \\
   a - s_3 &\leq 0 \\
   a - s_5 &\leq 0 \\
   z - a &\leq 15 \\
   l_1 &\ll 5 \iff s_1 - s_5 \leq -2 \\
   l_2 &\ll 4 \iff s_2 - s_4 \leq -5 \\
   l_3 &\ll 6 \iff s_3 - s_6 \leq -6 \\
   l_4 &\ll 2 \iff s_4 - s_2 \leq -4 \\
   l_5 &\ll 1 \iff s_5 - s_1 \leq -2 \\
   l_6 &\ll 3 \iff s_6 - s_3 \leq -3 \\
\end{align*}
\]
**Example: Jobshop Scheduling**

**T-Solver view**

- $s_1 - s_2 \leq -2$
- $z - a \leq 15$
- $s_3 - s_4 \leq -4$
- $l_{1 \prec 5} \iff s_1 - s_5 \leq -2$
- $s_5 - s_6 \leq -5$
- $l_{5 \prec 1} \iff s_5 - s_1 \leq -5$
- $s_2 - z \leq -5$
- $l_{2 \prec 4} \iff s_2 - s_4 \leq -5$
- $s_4 - z \leq -4$
- $l_{3 \prec 6} \iff s_3 - s_6 \leq -6$
- $s_6 - z \leq -3$
- $l_{6 \prec 3} \iff s_6 - s_3 \leq -3$
- $a - s_1 \leq 0$
- $l_{4 \prec 2} \iff s_4 - s_2 \leq -4$
- $a - s_3 \leq 0$
- $a - s_5 \leq 0$
Example: Jobshop Scheduling

**T-Solver view**

\[
\begin{align*}
    s_1 - s_2 &\leq -2 & z - a &\leq 15 \\
    s_3 - s_4 &\leq -4 & l_1 \prec 5 &\iff s_1 - s_5 \leq -2 \\
    s_5 - s_6 &\leq -5 & l_5 \prec 1 &\iff s_5 - s_1 \leq -5 \\
    s_2 - z &\leq -5 & l_2 \prec 4 &\iff s_2 - s_4 \leq -5 \\
    s_4 - z &\leq -4 & l_4 \prec 2 &\iff s_4 - s_2 \leq -4 \\
    s_6 - z &\leq -3 & l_3 \prec 6 &\iff s_3 - s_6 \leq -6 \\
    a - s_1 &\leq 0 & l_6 \prec 3 &\iff s_6 - s_3 \leq -3 \\
    a - s_3 &\leq 0 & & & \\
    a - s_5 &\leq 0 & & & \\
\end{align*}
\]

**CDCL-Solver view**

\[
\begin{align*}
    l_1 \prec 5 &\lor l_5 \prec 1 \\
    l_2 \prec 4 &\lor l_4 \prec 2 \\
    l_3 \prec 6 &\lor l_6 \prec 3 \\
\end{align*}
\]
Example: Jobshop Scheduling

- Reasoning: detection of negative cycles ([Bellman-Ford])

![Diagram with arrows and weights]

- $l_1 \prec 5 \iff s_1 - s_5 \leq -2$
- $l_5 \prec 1 \iff s_5 - s_1 \leq -5$
- $l_2 \prec 4 \iff s_2 - s_4 \leq -5$
- $l_4 \prec 2 \iff s_4 - s_2 \leq -4$
- $l_3 \prec 6 \iff s_3 - s_6 \leq -6$
- $l_6 \prec 3 \iff s_6 - s_3 \leq -3$
Example: Jobshop Scheduling

- **Reasoning:** detection of negative cycles ([Bellman-Ford])

![Graph with nodes and edges labeled with values]

\[ l_{1 < 5} \iff s_1 - s_5 \leq -2 \]
\[ l_{5 < 1} \iff s_5 - s_1 \leq -5 \]
\[ l_{2 < 4} \iff s_2 - s_4 \leq -5 \]
\[ l_{4 < 2} \iff s_4 - s_2 \leq -4 \]
\[ l_{3 < 6} \iff s_3 - s_6 \leq -6 \]
\[ l_{6 < 3} \iff s_6 - s_3 \leq -3 \]
Example: Jobshop Scheduling

Reasoning: detection of negative cycles ([Bellman-Ford])

\[ t_1 \approx t_2 \iff s_1 - s_5 \leq -2 \]
\[ t_5 \approx t_1 \iff s_5 - s_1 \leq -5 \]
\[ t_2 \approx t_4 \iff s_2 - s_4 \leq -5 \]
\[ t_4 \approx t_2 \iff s_4 - s_2 \leq -4 \]
\[ t_3 \approx t_6 \iff s_3 - s_6 \leq -6 \]
\[ t_6 \approx t_3 \iff s_6 - s_3 \leq -3 \]
Example: Jobshop Scheduling

- Reasoning: detection of negative cycles ([Bellman-Ford])

\[ l_{1<5} \iff s_1 - s_5 \leq -2 \]
\[ l_{5<1} \iff s_5 - s_1 \leq -5 \]
\[ l_{2<4} \iff s_2 - s_4 \leq -5 \]
\[ l_{4<2} \iff s_4 - s_2 \leq -4 \]
\[ l_{3<6} \iff s_3 - s_6 \leq -6 \]
\[ l_{6<3} \iff s_6 - s_3 \leq -3 \]
Example: Jobshop Scheduling

- Reasoning: detection of negative cycles ([Bellman-Ford])
Example: Jobshop Scheduling

- Reasoning: detection of negative cycles ([Bellman-Ford])

\[
\begin{align*}
&l_{1 < 5} \iff s_1 - s_5 \leq -2 \\
&l_{5 < 1} \iff s_5 - s_1 \leq -5 \\
&l_{2 < 4} \iff s_2 - s_4 \leq -5 \\
&l_{4 < 2} \iff s_4 - s_2 \leq -4 \\
&l_{3 < 6} \iff s_3 - s_6 \leq -6 \\
&l_{6 < 3} \iff s_6 - s_3 \leq -3
\end{align*}
\]
Example: Jobshop Scheduling

- Reasoning: detection of negative cycles ([Bellman-Ford])

\[
l_{1 < 5} \iff s_1 - s_5 \leq -2 \\
l_{5 < 1} \iff s_5 - s_1 \leq -5 \\
l_{2 < 4} \iff s_2 - s_4 \leq -5 \\
l_{4 < 2} \iff s_4 - s_2 \leq -4 \\
l_{3 < 6} \iff s_3 - s_6 \leq -6 \\
l_{6 < 3} \iff s_6 - s_3 \leq -3
\]
Example: Jobshop Scheduling

- Reasoning: detection of negative cycles ([Bellman-Ford])

Learned clause
- $l_{5<1} \lor l_{2<4}$
Results on Resource Constrained Project Scheduling Problem (RCPSP) [Ansótegui et al. 11]
Results on Resource Constrained Project Scheduling Problem (RCPSP) [Ansótegui et al. 11]

Two formulations

- Time encoding
- Task encoding
Results on Resource Constrained Project Scheduling Problem (RCPSP) [Ansótegui et al. 11]

Two formulations
- Time encoding
- Task encoding

Results
- More robust than lazy-FD
Results on Resource Constrained Project Scheduling Problem (RCPSP) [Ansótegui et al. 11]

**Two formulations**
- Time encoding
- Task encoding

**Results**
- More robust than lazy-FD
- State of the art for RCPSP!
Outline

1. Introduction
2. Scheduling and SAT Encoding
3. Scheduling and SAT Heuristics
4. Scheduling and SAT Hybrids
5. Conclusion
Conclusion
Conclusion

- Scheduling with SAT is not as bad as it sounds
Conclusion

- Scheduling with SAT is not as bad as it sounds
- Generic algorithms can sometimes be difficult to match
  - Adaptive heuristics
  - Clause learning

Nogood learning [Schiex & Verfaillie 93] and explanation for global constraints [Rochart & Jussien 03], disjunctive resource [Vilím 05]?

Hybridization (learning + dedicated reasoning) is the way to go

SAT Modulo Theories?

CDCL with global constraints and integer domains?

Explanation algorithms for global constraints?
Conclusion

- Scheduling with SAT is not as bad as it sounds
- Generic algorithms can sometimes be difficult to match
  - Adaptive heuristics
  - Clause learning
- Nogood learning [Schiex & Verfaillie 93] and explanation for global constraints [Rochart & Jussien 03], disjunctive resource [Vilím 05]?
  - Somehow it does not have the same impact as in SAT
Conclusion

- Scheduling with SAT is not as bad as it sounds
- Generic algorithms can sometimes be difficult to match
  - Adaptive heuristics
  - Clause learning
- Nogood learning [Schiex & Verfaillie 93] and explanation for global constraints [Rochart & Jussien 03], disjunctive resource [Vilím 05]? 
  - Somehow it does not have the same impact as in SAT
- Hybridization (learning + dedicated reasoning) is the way to go
  - SAT Modulo Theories?
  - CDCL with global constraints and integer domains?
  - Explanation algorithms for global constraints?