Representations for Automated Reasoning

Marijn Heule

Carnegie Mellon University

http://www.cs.cmu.edu/~mheule/15816-f24/

Automated Reasoning and Satisfiability
September 4, 2024

The Right Representation is Crucial

What makes some problems hard and others easy?

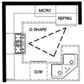
Does the representation enable efficient reasoning?

The famous pigeonhole principle

- Hard for many automated reasoning approaches
- Easy for a little kid given the right representation

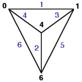
source: pecanpartnership.co.uk/2016/01/05/beware-pigeon-hole-overcoming-stereotypes-build-collaborative-culture

Encoding problems into SAT

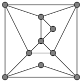


Architectural 3D Layout
[VSMM '07]
Henriette Bier

Edge-matching Puzzles
[LaSh '08]

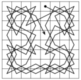


Graceful Graphs
[AAAI '10]
Toby Walsh

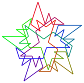


Clique-Width
[SAT '13, TOCL '15]
Stefan Szeider

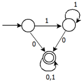
Firewall Verification
[SSS '16]
Mohamed Gouda



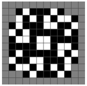
Open Knight Tours
Moshe Vardi



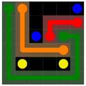
Van der Waerden numbers
[EJoC '07]



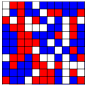
Software Model Synthesis
[ICGI '10, ESE '13]
Sicco Verwer



Conway's Game of Life
[EJoC '13]
Willem van der Poel



Connect the Pairs
Donald Knuth



Pythagorean Triples
[SAT '16, CACM '17]
Victor Marek

Collatz conjecture [Open]
Emre Yolcu
Scott Aaronson

Common Constraints

Tseitin Transformation

Representing Integers

Cardinality Constraints

Hamiltonian Cycles

Common Constraints

Tseitin Transformation

Representing Integers

Cardinality Constraints

Hamiltonian Cycles

Common Constraints: Consistency and Arc-Consistency

- Let us consider an encoding of a constraint C with a correspondence between assignments of the variables in C with Boolean assignments of the variables in the encoding
- The encoding is consistent if whenever is partial assignment inconsistent w.r.t. C (i.e., cannot be extended to a solution of C), unit propagation results in a conflict

Common Constraints: Consistency and Arc-Consistency

- Let us consider an encoding of a constraint C with a correspondence between assignments of the variables in C with Boolean assignments of the variables in the encoding
- The encoding is consistent if whenever is partial assignment inconsistent w.r.t. C (i.e., cannot be extended to a solution of C), unit propagation results in a conflict
- The encoding is arc-consistent if
  1. it is consistent, and
  2. unit propagation discards values that cannot be assigned
- These are good properties for encodings: SAT solvers are very good at unit propagation!

Common Constraints: AtMostOne

E.g. the AtMostOne constraint $x_1 + x_2 + \ldots + x_n \leq 1$:

- Consistency $\equiv$ if there are two variables x_i assigned to *true* then unit propagation should give a conflict

- Arc-consistency $\equiv$ Consistency + if there is one x_i assigned to *true* then all others x_j should be assigned to *false* by unit propagation

Common Constraints: AtMostOne

E.g. the AtMostOne constraint $x_1 + x_2 + \ldots + x_n \leq 1$:

- Consistency $\equiv$ if there are two variables x_i assigned to *true* then unit propagation should give a conflict
- Arc-consistency $\equiv$ Consistency + if there is one x_i assigned to *true* then all others x_j should be assigned to *false* by unit propagation

The direct encoding requires $n(n-1)/2$ binary clauses:

$$\bigwedge_{1 \leq i < j \leq n} (\overline{x}_i \vee \overline{x}_j)$$

Is it possible to use fewer clauses?

Common Constraints: Compact AtMostOne

Given a set of Boolean variables $x_1, \ldots, x_n$, how to encode

$$\textsc{AtMostOne}\,(x_1, \ldots, x_n)$$

into SAT using a linear number of binary clauses?

Common Constraints: Compact AtMostOne

Given a set of Boolean variables $x_1, \ldots, x_n$, how to encode

$$\textsc{AtMostOne}\,(x_1, \ldots, x_n)$$

into SAT using a linear number of binary clauses?

Use auxiliary variables: Apply the direct encoding if $n \leq 4$ otherwise replace $\textsc{AtMostOne}\,(x_1, \ldots, x_n)$ by

$$\textsc{AtMostOne}\,(x_1, x_2, x_3, \overline{y}) \land \textsc{AtMostOne}\,(y, x_4, \ldots, x_n)$$

resulting in $3n - 6$ clauses and $(n - 3)/2$ new variables

Note: $\textsc{AtMostOne}\,(x_1, x_2, x_3, \overline{y}) \equiv$
$\textsc{AtMostOne}\,(x_1, x_2, x_3) \land (\overline{x}_1 \lor y) \land (\overline{x}_2 \lor y) \land (\overline{x}_3 \lor y) \equiv$
$\textsc{AtMostOne}\,(x_1, x_2, x_3) \land (x_1 \lor x_2 \lor x_3) \rightarrow y$

Common Constraints: Non-Consistent AtLeastOne

Given a set of Boolean variables $x_1, \ldots, x_n$, how to encode

$$x_1 + \cdots + x_n \geq 1$$

into SAT?

Hint: This is easy...

Common Constraints: Non-Consistent AtLeastOne

Given a set of Boolean variables $x_1, \ldots, x_n$, how to encode

$$x_1 + \cdots + x_n \geq 1$$

into SAT?

Hint: This is easy...

$$(x_1 \vee x_2 \vee \cdots \vee x_n)$$

Common Constraints: Non-Consistent AtLeastOne

Given a set of Boolean variables $x_1, \ldots, x_n$, how to encode

$$x_1 + \cdots + x_n \geq 1$$

into SAT?

Hint: This is easy...

$$(x_1 \vee x_2 \vee \cdots \vee x_n)$$

Example

The following encoding of $x_1 + x_2 + x_3 + x_4 \geq 1$ is not consistent (using auxiliary variables y and z):

$$(y \vee z \vee x_1) \wedge (y \vee \overline{z} \vee x_2) \wedge (\overline{y} \vee z \vee x_3) \wedge (\overline{y} \vee \overline{z} \vee x_4)$$

Common Constraints: Exclusive OR

Given a set of Boolean variables $x_1, \ldots, x_n$, how to encode

$$\mathrm{XOR}(x_1, \ldots, x_n)$$

into SAT using a linear number of binary clauses?

The direct encoding requires 2^{n-1} clauses of length n:

$$\bigwedge_{\text{even } \#\neg} (\overline{x}_1 \vee \overline{x}_2 \vee \cdots \vee \overline{x}_n)$$

Common Constraints: Exclusive OR

Given a set of Boolean variables $x_1, \ldots, x_n$, how to encode

$$\mathrm{XOR}(x_1, \ldots, x_n)$$

into SAT using a linear number of binary clauses?

The direct encoding requires 2^{n-1} clauses of length n:

$$\bigwedge_{\text{even } \#\neg} (\overline{x}_1 \vee \overline{x}_2 \vee \cdots \vee \overline{x}_n)$$

Make it compact: $\mathrm{XOR}\,(x_1, x_2, x_3, \overline{y}) \wedge \mathrm{XOR}\,(y, x_4, \ldots, x_n)$

Note: $\mathrm{XOR}\,(x_1, x_2, x_3, \overline{y}) \equiv y \leftrightarrow \mathrm{XOR}\,(x_1, x_2, x_3)$

Tradeoff: more variables but fewer clauses!

Common Constraints: Linear versus Pooled

Details regarding splitting can impact the performance

Linear encoding with cutoff k:

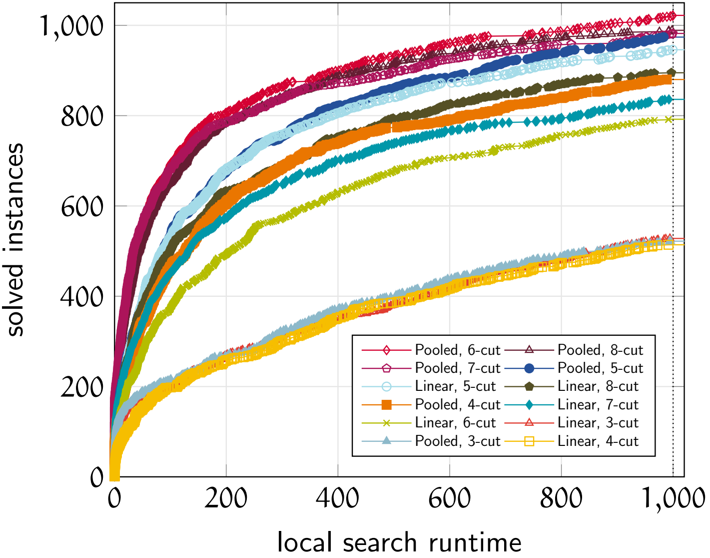
- XOR $(x_1, \ldots, x_k, \overline{y}) \wedge$ XOR $(y, x_{k+1}, \ldots, x_n)$

Pooled encoding with cutoff k:

- XOR $(x_1, \ldots, x_k, \overline{y}) \wedge$ XOR $(x_{k+1}, \ldots, x_n, y)$

I always use the pooled encoding, e.g. for matrix multiplication instances [SAT'19], as it appears more effective

Common Constraints: Impact on Matrix Multiplication



AtMostOne: Satisfiability Equivalence

Are these two encoding of $\textsc{AtMostOne}(x_1, x_2)$ equivalent?

F_1 (direct encoding)	F_2 (split encoding)
$\overline{x}_1 \vee \overline{x}_2$	$\overline{x}_1 \vee y$
	$\overline{y} \vee \overline{x}_2$

Question: Is F_1 equivalent to F_2?

Note: $F_1 \leftrightarrow F_2$ is valid if $\neg F_1 \wedge F_2$ and $F_1 \wedge \neg F_2$ are unsatisfiable.

AtMostOne: Satisfiability Equivalence

Are these two encoding of $\textsc{AtMostOne}(x_1, x_2)$ equivalent?

F_1 (direct encoding)	F_2 (split encoding)
$\overline{x}_1 \vee \overline{x}_2$	$\overline{x}_1 \vee y$
	$\overline{y} \vee \overline{x}_2$

Is $\neg F_1 \wedge F_2$ unsatisfiable?

Note: $\neg F_1 \equiv x_1 \wedge x_2$

AtMostOne: Satisfiability Equivalence

Are these two encoding of $\textsc{AtMostOne}(x_1, x_2)$ equivalent?

F_1 (direct encoding)	F_2 (split encoding)
$\overline{x}_1 \vee \overline{x}_2$	$\overline{x}_1 \vee y$
	$\overline{y} \vee \overline{x}_2$

Is $\neg F_1 \wedge F_2$ unsatisfiable? yes!

Note: $\neg F_1 \equiv x_1 \wedge x_2$

AtMostOne: Satisfiability Equivalence

Are these two encoding of $\textsc{AtMostOne}(x_1, x_2)$ equivalent?

F_1 (direct encoding)	F_2 (split encoding)
$\overline{x}_1 \vee \overline{x}_2$	$\overline{x}_1 \vee y$
	$\overline{y} \vee \overline{x}_2$

Is $F_1 \wedge \neg F_2$ unsatisfiable?

Note: $\neg F_2 \equiv \overline{(\overline{x}_1 \vee y) \wedge (\overline{y} \vee \overline{x}_2)} \equiv (x_1 \wedge \overline{y}) \vee (y \wedge x_2)$
$\equiv (x_1 \vee y) \wedge (x_1 \vee x_2) \wedge (\overline{y} \vee x_2)$

AtMostOne: Satisfiability Equivalence

Are these two encoding of $\textsc{AtMostOne}(x_1, x_2)$ equivalent?

F_1 (direct encoding)	F_2 (split encoding)
$\overline{x}_1 \vee \overline{x}_2$	$\overline{x}_1 \vee y$
	$\overline{y} \vee \overline{x}_2$

Is $F_1 \wedge \neg F_2$ unsatisfiable? no!

Note: $\neg F_2 \equiv \overline{(\overline{x}_1 \vee y) \wedge (\overline{y} \vee \overline{x}_2)} \equiv (x_1 \wedge \overline{y}) \vee (y \wedge x_2)$
$\equiv (x_1 \vee y) \wedge (x_1 \vee x_2) \wedge (\overline{y} \vee x_2)$

AtMostOne: Satisfiability Equivalence

Are these two encoding of $\textsc{AtMostOne}(x_1, x_2)$ equivalent?

F_1 (direct encoding)	F_2 (split encoding)
$\overline{x}_1 \vee \overline{x}_2$	$\overline{x}_1 \vee y$
	$\overline{y} \vee \overline{x}_2$

F_1 and F_2 are equisatisfiable:

- F_1 is satisfiable iff F_2 is satisfiable.

Note: Equisatisfiability is weaker than equivalence but useful if all we want we want to do is determine satisfiability.

Common Constraints

Tseitin Transformation

Representing Integers

Cardinality Constraints

Hamiltonian Cycles

Tseitin Transformation: Negation Normal Form

The set of propositional formulas in negation normal form (NNF) is generated inductively as follows:

- Each propositional variable p and the negation $\overline{p}$ of a propositional variable are in negation normal form
- If A and B are in negation normal form, then so are $A \wedge B$ and $A \vee B$

Example $\big((p \wedge q \wedge \overline{r}) \vee (r \wedge (\overline{p} \vee \overline{q}))\big) \wedge (\overline{s} \vee (p \wedge t))$

Tseitin Transformation: Negation Normal Form

The set of propositional formulas in negation normal form (NNF) is generated inductively as follows:

- Each propositional variable p and the negation $\overline{p}$ of a propositional variable are in negation normal form
- If A and B are in negation normal form, then so are $A \wedge B$ and $A \vee B$

Example $\big((p \wedge q \wedge \overline{r}) \vee (r \wedge (\overline{p} \vee \overline{q}))\big) \wedge (\overline{s} \vee (p \wedge t))$

Propositional formulas can be converted into NNF using:

- $A \rightarrow B \equiv \overline{A} \vee B$
- $\overline{(A \vee B)} \equiv (\overline{A} \wedge \overline{B})$
- $\overline{(A \wedge B)} \equiv (\overline{A} \vee \overline{B})$
- $\overline{\overline{A}} \equiv A$

Tseitin Transformation: Avoid Exponential Blowup

What is the complexity of transformation NNF into CNF?

- $A \wedge (B \vee C) \equiv (A \wedge B) \vee (A \wedge C)$
- $A \vee (B \wedge C) \equiv (A \vee B) \wedge (A \vee C)$

Tseitin Transformation: Avoid Exponential Blowup

What is the complexity of transformation NNF into CNF?

- $A \wedge (B \vee C) \equiv (A \wedge B) \vee (A \wedge C)$
- $A \vee (B \wedge C) \equiv (A \vee B) \wedge (A \vee C)$

In some cases, converting NNF to CNF can have an exponential explosion on the size of the formula.

If we convert the NNF $(x_1 \wedge y_1) \vee (x_2 \wedge y_2) \vee \ldots \vee (x_n \wedge y_n)$ using the above distributive laws into CNF:

$(x_1 \vee x_2 \vee \ldots \vee x_n) \wedge (y_1 \vee x_2 \vee \ldots \vee x_n) \wedge \ldots \wedge (y_1 \vee y_2 \vee \ldots \vee y_n)$

- How can we avoid the exponential blowup? In this case, the equivalent formula would have 2^n clauses!

Tseitin Transformation: Avoid Exponential Blowup

What is the complexity of transformation NNF into CNF?

- $A \wedge (B \vee C) \equiv (A \wedge B) \vee (A \wedge C)$
- $A \vee (B \wedge C) \equiv (A \vee B) \wedge (A \vee C)$

In some cases, converting NNF to CNF can have an exponential explosion on the size of the formula.

If we convert the NNF $(x_1 \wedge y_1) \vee (x_2 \wedge y_2) \vee \ldots \vee (x_n \wedge y_n)$ using the above distributive laws into CNF:

$$(x_1 \vee x_2 \vee \ldots \vee x_n) \wedge (y_1 \vee x_2 \vee \ldots \vee x_n) \wedge \ldots \wedge (y_1 \vee y_2 \vee \ldots \vee y_n)$$

- How can we avoid the exponential blowup? In this case, the equivalent formula would have 2^n clauses!

Tseitin's transformation converts a formula F into an equisatisfiable CNF formula that is linear in the size of F!

- Key idea: introduce auxiliary variables to represent subformulas, and define those variables using CNF clauses

Tseitin: Small Example

Consider the formula $F = p \vee (q \wedge r)$

Tseitin: Small Example

Consider the formula $F = p \lor (q \land r)$

We can add the definition $d \leftrightarrow (q \land r)$

Replacing $(q \land r)$ by d results in CNF $p \lor d$

Tseitin: Small Example

Consider the formula $F = p \lor (q \land r)$

We can add the definition $d \leftrightarrow (q \land r)$

Replacing $(q \land r)$ by d results in CNF $p \lor d$

The clauses representing the definition are:

$$(\neg d \lor q) \land (\neg d \lor r) \land (d \lor \neg q \lor \neg r)$$

An equisatisfiable formula of F in CNF is:

$$(p \lor d) \land (\neg d \lor q) \land (\neg d \lor r) \land (d \lor \neg q \lor \neg r)$$

Satisfying the resulting formula satisfies F on original variables

Tseitin: A Linear-Size Transformation

Why is the Tseitin transformation interesting?

- At most a linear number of definitions
- Definitions can be easily converted into clauses
- Easily obtain a satisfying assignment for original formula
- Resulting in an efficient transformation into CNF

Tseitin: Implementation and Optimizations

Implementation:

- Convert the formula into NNF (not necessary, good practice)
- Generate the definitions from left to right
- OR definition: $d \leftrightarrow x_1 \vee x_2 \vee \cdots \vee x_k \equiv$ $(x_1 \vee x_2 \vee \cdots \vee x_k \vee \overline{d}) \wedge (\overline{x}_1 \vee d) \wedge (\overline{x}_2 \vee d) \wedge \cdots \wedge (\overline{x}_k \vee d)$
- AND definition: $d \leftrightarrow x_1 \wedge x_2 \wedge \cdots \wedge x_k \equiv$ $(\overline{x}_1 \vee \overline{x}_2 \vee \cdots \vee \overline{x}_k \vee d) \wedge (x_1 \vee \overline{d}) \wedge (x_2 \vee \overline{d}) \wedge \cdots \wedge (x_k \vee \overline{d})$

Tseitin: Implementation and Optimizations

Implementation:

- Convert the formula into NNF (not necessary, good practice)
- Generate the definitions from left to right
- OR definition: $d \leftrightarrow x_1 \lor x_2 \lor \cdots \lor x_k \equiv$ $(x_1 \lor x_2 \lor \cdots \lor x_k \lor \overline{d}) \land (\overline{x}_1 \lor d) \land (\overline{x}_2 \lor d) \land \cdots \land (\overline{x}_k \lor d)$
- AND definition: $d \leftrightarrow x_1 \land x_2 \land \cdots \land x_k \equiv$ $(\overline{x}_1 \lor \overline{x}_2 \lor \cdots \lor \overline{x}_k \lor d) \land (x_1 \lor \overline{d}) \land (x_2 \lor \overline{d}) \land \cdots \land (x_k \lor \overline{d})$

Optimizations:

- Reuse definitions when possible
- Avoid definitions by interpreting an NNF formula as a CNF formula: e.g. $p \lor (q \land \neg r) \lor \neg s$

Tseitin: Larger Example without Optimization

Convert the following NNF into CNF:

$$\big((p \land q \land \overline{r}) \lor (r \land (\overline{p} \lor \overline{q}))\big) \land (\overline{s} \lor (p \land t))$$

Which results in the following definitions:

- $d_1 \leftrightarrow p \land q \land \overline{r}$

Tseitin: Larger Example without Optimization

Convert the following NNF into CNF:

$$\big((p \land q \land \overline{r}) \lor (r \land (\overline{p} \lor \overline{q}))\big) \land (\overline{s} \lor (p \land t))$$

Which results in the following definitions:

- $d_1 \leftrightarrow p \land q \land \overline{r}$
- $d_2 \leftrightarrow \overline{p} \lor \overline{q}$

Tseitin: Larger Example without Optimization

Convert the following NNF into CNF:

$$\big((p \land q \land \overline{r}) \lor (r \land (\overline{p} \lor \overline{q}))\big) \land (\overline{s} \lor (p \land t))$$

Which results in the following definitions:

- $d_1 \leftrightarrow p \land q \land \overline{r}$
- $d_2 \leftrightarrow \overline{p} \lor \overline{q}$
- $d_3 \leftrightarrow r \land d_2$

Tseitin: Larger Example without Optimization

Convert the following NNF into CNF:

$$\big((p \land q \land \overline{r}) \lor (r \land (\overline{p} \lor \overline{q}))\big) \land (\overline{s} \lor (p \land t))$$

Which results in the following definitions:

- $d_1 \leftrightarrow p \land q \land \overline{r}$
- $d_2 \leftrightarrow \overline{p} \lor \overline{q}$
- $d_3 \leftrightarrow r \land d_2$
- $d_4 \leftrightarrow d_1 \lor d_3$

Tseitin: Larger Example without Optimization

Convert the following NNF into CNF:

$$\big((p \land q \land \overline{r}) \lor (r \land (\overline{p} \lor \overline{q}))\big) \land (\overline{s} \lor (p \land t))$$

Which results in the following definitions:

- $d_1 \leftrightarrow p \land q \land \overline{r}$
- $d_2 \leftrightarrow \overline{p} \lor \overline{q}$
- $d_3 \leftrightarrow r \land d_2$
- $d_4 \leftrightarrow d_1 \lor d_3$
- $d_5 \leftrightarrow p \land t$

Tseitin: Larger Example without Optimization

Convert the following NNF into CNF:

$$\big((p \land q \land \overline{r}) \lor (r \land (\overline{p} \lor \overline{q}))\big) \land (\overline{s} \lor (p \land t))$$

Which results in the following definitions:

- $d_1 \leftrightarrow p \land q \land \overline{r}$
- $d_2 \leftrightarrow \overline{p} \lor \overline{q}$
- $d_3 \leftrightarrow r \land d_2$
- $d_4 \leftrightarrow d_1 \lor d_3$
- $d_5 \leftrightarrow p \land t$
- $d_6 \leftrightarrow \overline{s} \lor d_5$

Tseitin: Larger Example without Optimization

Convert the following NNF into CNF:

$$\big((p \land q \land \overline{r}) \lor (r \land (\overline{p} \lor \overline{q}))\big) \land (\overline{s} \lor (p \land t))$$

Which results in the following definitions:

- $d_1 \leftrightarrow p \land q \land \overline{r}$
- $d_2 \leftrightarrow \overline{p} \lor \overline{q}$
- $d_3 \leftrightarrow r \land d_2$
- $d_4 \leftrightarrow d_1 \lor d_3$
- $d_5 \leftrightarrow p \land t$
- $d_6 \leftrightarrow \overline{s} \lor d_5$
- $d_7 \leftrightarrow d_4 \land d_6$

Tseitin: Larger Example with Optimization

Convert the following NNF into CNF:

$$\big((p \land q \land \overline{r}) \lor (r \land (\overline{p} \lor \overline{q}))\big) \land (\overline{s} \lor (p \land t))$$

Which results in the following definitions:

- $d_1 \leftrightarrow p \land q \land \overline{r}$

Tseitin: Larger Example with Optimization

Convert the following NNF into CNF:

$$\big((p \land q \land \overline{r}) \lor (r \land (\overline{p} \lor \overline{q}))\big) \land (\overline{s} \lor (p \land t))$$

Which results in the following definitions:

- $d_1 \leftrightarrow p \land q \land \overline{r}$
- $d_2 \leftrightarrow \neg p \lor \neg q$

Tseitin: Larger Example with Optimization

Convert the following NNF into CNF:

$$\big((p \land q \land \overline{r}) \lor (r \land (\overline{p} \lor \overline{q}))\big) \land (\overline{s} \lor (p \land t))$$

Which results in the following definitions:

- $d_1 \leftrightarrow p \land q \land \overline{r}$
- $d_2 \leftrightarrow \neg p \lor \neg q$
- $d_3 \leftrightarrow r \land d_2$

Tseitin: Larger Example with Optimization

Convert the following NNF into CNF:

$$\big((p \land q \land \bar{r}) \lor (r \land (\bar{p} \lor \bar{q}))\big) \land (\bar{s} \lor (p \land t))$$

Which results in the following definitions:

- $d_1 \leftrightarrow p \land q \land \bar{r}$
- $d_2 \leftrightarrow \neg p \lor \neg q$
- $d_3 \leftrightarrow r \land d_2$
- $d_4 \leftrightarrow p \land t$

Tseitin: Larger Example with Optimization

Convert the following NNF into CNF:

$$\big((p \land q \land \overline{r}) \lor (r \land (\overline{p} \lor \overline{q}))\big) \land (\overline{s} \lor (p \land t))$$

Which results in the following definitions:

- $d_1 \leftrightarrow p \land q \land \overline{r}$
- $d_2 \leftrightarrow \neg p \lor \neg q$
- $d_3 \leftrightarrow r \land d_2$
- $d_4 \leftrightarrow p \land t$

Final result: $(d_1 \lor d_3) \land (\overline{s} \lor d_4)$ plus definition clauses

Tseitin Transformation: Automated Tools

- Using automated tools to encode to CNF:
  limboole: `http://fmv.jku.at/limboole`

- Tseitin's encoding may add many redundant variables/clauses!
- Using **limboole** for the pigeon hole problem ($n = 3$) creates a formula with 40 variables and 98 clauses
- After unit propagation the formula has 12 variables and 28 clauses
- Original CNF formula only has 6 variables and 9 clauses

Common Constraints

Tseitin Transformation

Representing Integers

Cardinality Constraints

Hamiltonian Cycles

Representing Integers: Direct Encoding

- Each number i is represented by a Boolean variable: d_i
- At least one number is true: $d_0 \vee \cdots \vee d_n$
- At most one number is true: $\bigwedge_{i<j} \overline{d}_i \vee \overline{d}_j$
- Expressing in a clause that an integer has a specific value v requires one literal.
- For example, "if the number is 1, then do x", is encoded as $\overline{d}_1 \vee x$.
- Typically effective when reasoning about a small range of integers.

Representing Integers: Order Encoding

Order encoding:

- Variables represent that a number is larger or equal: $o_{\geq i}$
- Requires a linear number of binary clauses: $o_{\geq i} \vee \overline{o}_{\geq i+1}$
- Expressing in a clause that an integer has a specific value v requires two literals.
- For example, "if the number is 1, then do x", is encoded as $\overline{o}_{\geq 1} \vee o_{\geq 2} \vee x$.
- Allows the solver to reason (and produce clauses) that cover multiple cases.

Representing Integers: Binary Encoding

Binary encoding:

- Use $\lceil \log_2 n \rceil$ auxiliary variables b_i to represent n in binary
- All non-occurring numbers $\leq 2^{\lceil \log_2 n \rceil}$ need to be blocked. For example, if we have the numbers 0, 1, and 2, then the number 3 needs to be blocked: $(\neg b_0 \vee \neg b_1)$
- Expressing in a clause that an integer has a specific value v requires $\lceil \log_2 n \rceil$ literals.
- For example, "if the number is 1, then do x", is encoded as $\neg b_0 \vee b_1 \vee x$.
- Typically effective when reasoning about a large range of integers.

Common Constraints

Tseitin Transformation

Representing Integers

Cardinality Constraints

Hamiltonian Cycles

Cardinality Constraints: AtMostOne

Recall AtMostOne constraints:

- AtMostOne: $x_1 + x_2 + x_3 + x_4 \leq 1$
- Clauses for the naive (or direct) encoding:

$$\left.\begin{array}{c} (x_1 \rightarrow \overline{x}_2) \\ (x_1 \rightarrow \overline{x}_3) \\ (x_1 \rightarrow \overline{x}_4) \\ \ldots \end{array}\right\} \begin{array}{c} \overline{x}_1 \vee \overline{x}_2 \\ \overline{x}_1 \vee \overline{x}_3 \\ \overline{x}_1 \vee \overline{x}_4 \\ \ldots \end{array}$$

- Complexity: $\binom{n}{2}$ or $\mathcal{O}(n^2)$ clauses

This can be reduced to a linear number using auxiliary variables

What about the general case for cardinality constraints?

Cardinality Constraints: AtMostK

AtMostK constraints:

- General constraint: $x_1 + \cdots + x_n \leq k$
- Example constraint: $x_1 + x_2 + x_3 + x_4 \leq 2$
- Clauses for the naive encoding:

$$\left.\begin{array}{c} (x_1 \wedge x_2 \rightarrow \overline{x}_3) \\ (x_1 \wedge x_2 \rightarrow \overline{x}_4) \\ (x_2 \wedge x_3 \rightarrow \overline{x}_4) \\ \dots \end{array}\right\} \begin{array}{c} (\overline{x}_1 \vee \overline{x}_2 \vee \overline{x}_3) \\ (\overline{x}_1 \vee \overline{x}_2 \vee \overline{x}_4) \\ (\overline{x}_2 \vee \overline{x}_3 \vee \overline{x}_4) \\ \dots \end{array}$$

- Complexity: $\binom{n}{k+1}$ or $\mathcal{O}(n^k)$ clauses

Cardinality Constraints: AtMostK

AtMostK constraints:

- General constraint: $x_1 + \cdots + x_n \leq k$
- Example constraint: $x_1 + x_2 + x_3 + x_4 \leq 2$
- Clauses for the naive encoding:

$$\left.\begin{array}{c}(x_1 \wedge x_2 \rightarrow \overline{x}_3)\\(x_1 \wedge x_2 \rightarrow \overline{x}_4)\\(x_2 \wedge x_3 \rightarrow \overline{x}_4)\\\ldots\end{array}\right\}\begin{array}{c}(\overline{x}_1 \vee \overline{x}_2 \vee \overline{x}_3)\\(\overline{x}_1 \vee \overline{x}_2 \vee \overline{x}_4)\\(\overline{x}_2 \vee \overline{x}_3 \vee \overline{x}_4)\\\ldots\end{array}$$

- Complexity: $\binom{n}{k+1}$ or $\mathcal{O}(n^k)$ clauses

Can we build an encoding that is arc-consistent and uses a polynomial number of clauses for at-most-k constraints?

Cardinality Constraints: AtMostK

AtMostK constraints:

- General constraint: $x_1 + \cdots + x_n \leq k$
- Example constraint: $x_1 + x_2 + x_3 + x_4 \leq 2$
- Clauses for the naive encoding:

$$\left.\begin{array}{c} (x_1 \wedge x_2 \rightarrow \overline{x}_3) \\ (x_1 \wedge x_2 \rightarrow \overline{x}_4) \\ (x_2 \wedge x_3 \rightarrow \overline{x}_4) \\ \dots \end{array}\right\} \begin{array}{c} (\overline{x}_1 \vee \overline{x}_2 \vee \overline{x}_3) \\ (\overline{x}_1 \vee \overline{x}_2 \vee \overline{x}_4) \\ (\overline{x}_2 \vee \overline{x}_3 \vee \overline{x}_4) \\ \dots \end{array}$$

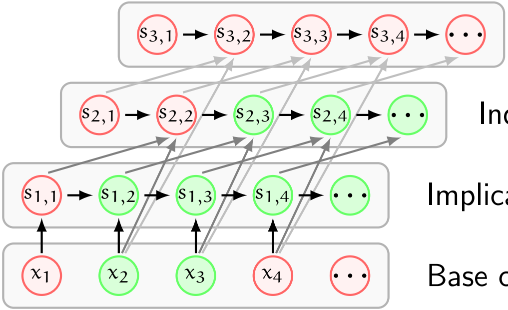
- Complexity: $\binom{n}{k+1}$ or $\mathcal{O}(n^k)$ clauses

Can we build an encoding that is arc-consistent and uses a polynomial number of clauses for at-most-k constraints?

Yes! With $O(n \cdot k)$ auxiliary variables, we need $O(n \cdot k)$ clauses!

Cardinality Constraints: Sinz Encoding

Introduce auxiliary variables $s_{i,j}$ with the following meaning: the sum of the first j literals is larger or equal to i



Constraint: $\overline{s}_{k+1,n}$

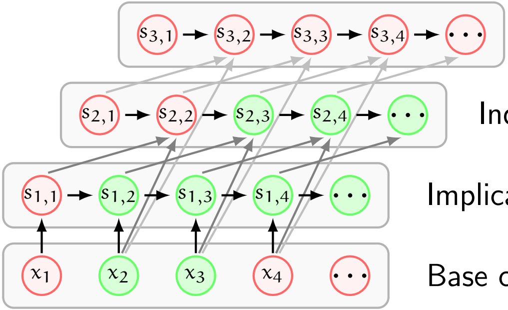
Ind case: $\overline{x}_{j+1} \vee \overline{s}_{i,j} \vee s_{i+1,j+1}$

Implication: $\overline{s}_{i,j} \vee s_{i,j+1}$

Base case: $\overline{x}_j \vee s_{1,j}$

Cardinality Constraints: Sinz Encoding

Introduce auxiliary variables $s_{i,j}$ with the following meaning: the sum of the first j literals is larger or equal to i



Constraint: $\overline{s}_{k+1,n}$

Ind case: $\overline{x}_{j+1} \vee \overline{s}_{i,j} \vee s_{i+1,j+1}$

Implication: $\overline{s}_{i,j} \vee s_{i,j+1}$

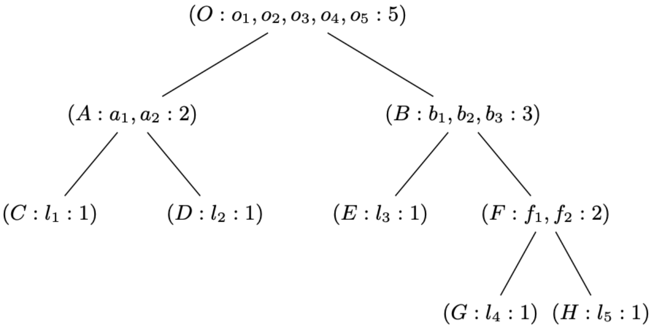
Base case: $\overline{x}_j \vee s_{1,j}$

- Arc-consistent using $\mathcal{O}(n \cdot k)$ variables and clauses
- More details in paper: "Towards an Optimal CNF Encoding of Boolean Cardinality Constraints", CP2005

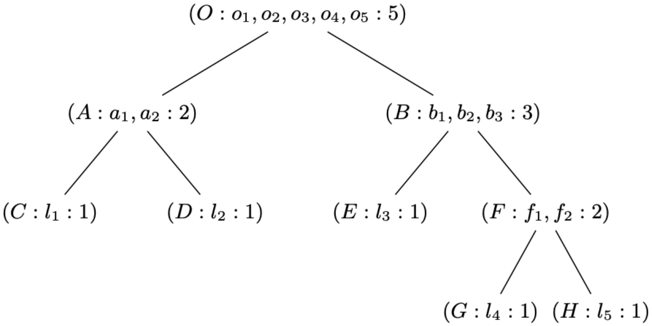
Cardinality Constraints: Totalizer encoding (1)

What is another example of an at-most-k encoding for $l_1 + \ldots l_5 \leq k$?

Totalizer encoding is based on a tree structure and also only needs $O(n \cdot k)$ clauses/variables.



Cardinality Constraints: Totalizer encoding (2)

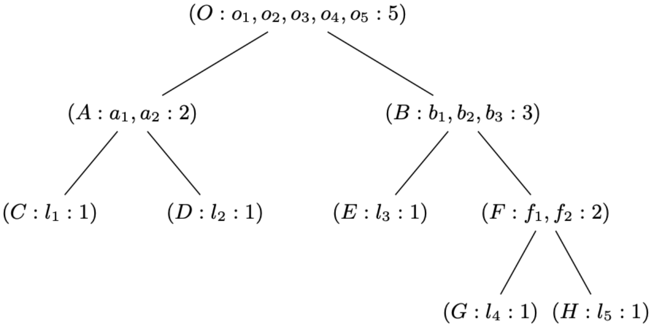


Use auxiliary variables to count the sum of the subtree:

- $f_1 \equiv l_4 + l_5 = 1$
- $f_2 \equiv l_4 + l_5 = 2$

Note that only f_1 or f_2 will be assigned to 1.

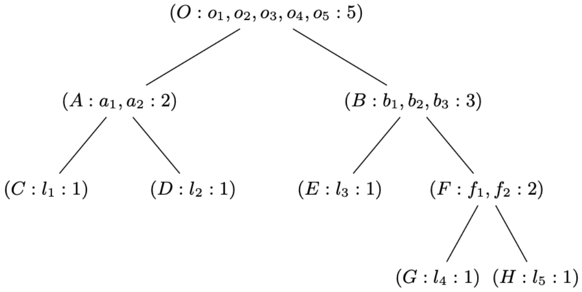
Cardinality Constraints: Totalizer encoding (3)



Use auxiliary variables to count the sum of the subtree:

- $b_1 \equiv l_3 + f_1 + 2 \times f_2 = 1$
- $b_2 \equiv l_3 + f_1 + 2 \times f_2 = 2$
- $b_3 \equiv l_3 + f_1 + 2 \times f_2 = 3$

Cardinality Constraints: Totalizer encoding (4)



Any parent node P, counting up to n_P, has two children L and R counting up to n_L and n_R respectively s.t. $n_L + n_R = n_P$.

Common Constraints

Tseitin Transformation

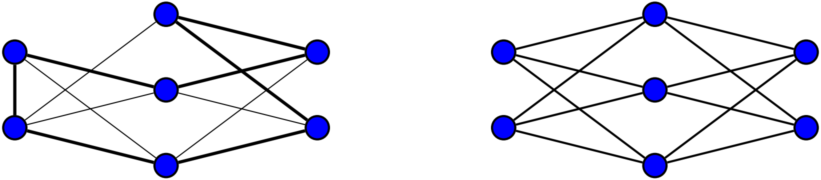
Representing Integers

Cardinality Constraints

Hamiltonian Cycles

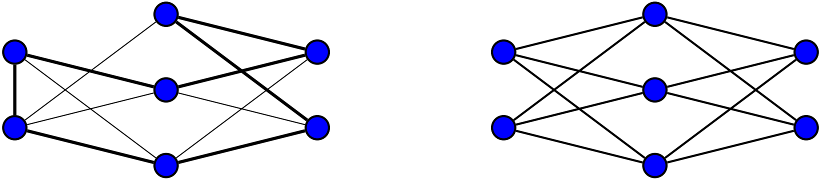
Hamiltonian Cycles: Two Constraints

Hamiltonian Cycle Problem (HCP):
Does there exist a cycle that visits all vertices exactly once?



Hamiltonian Cycles: Two Constraints

Hamiltonian Cycle Problem (HCP):
Does there exist a cycle that visits all vertices exactly once?

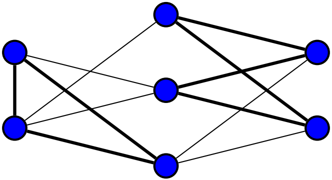


Two constraints:

- Exactly two edges per vertex: easy cardinality constraints
- Exactly one cycle: hard to be compact and arc-consistent
  - One option is to ignore the constraint: lazy encoding
  - Various encodings use $O(|V|^3)$. Too large for many graphs
  - For large graphs we need encodings that are quasi-linear in $|E|$

Hamiltonian Cycles: Lazy Encoding

Hamiltonian Cycle Problem (HCP):
Does there exist a cycle that visits all vertices exactly once?

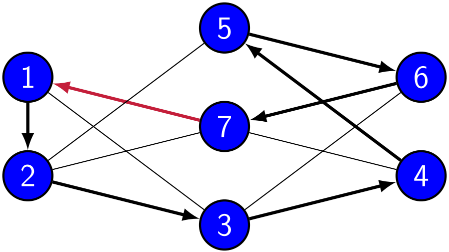


Only encode the two-edges-per-vertex constraint

If a solution has multiple cycles: block the smallest one

- Use incremental SAT to keep conflict clauses
- SMT is based on a similar approach

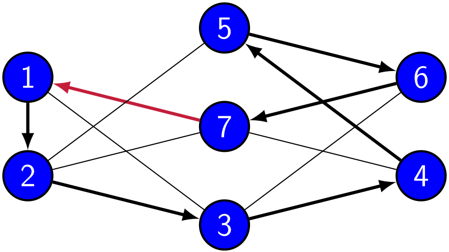
Hamiltonian Cycles: Encodings Quasi-Linear in |E|



Key elements:

- Each vertex have an index in the range $\{1, \ldots, |V|\}$.
- Selected edges are directed.
- Each vertex has one incoming and one outgoing edge.
- For each directed edge (u, v): the index of v is the successor of the index of u — except for the starting vertex.

Hamiltonian Cycles: Encodings Quasi-Linear in |E|



Key elements:

- Each vertex have an index in the range $\{1, \ldots, |V|\}$.
- Selected edges are directed.
- Each vertex has one incoming and one outgoing edge.
- For each directed edge (u, v): the index of v is the successor of the index of u — except for the starting vertex.

How to implement the successor property?

Hamiltonian Cycles: Binary Adder Encoding [Zhou 2020]

Each index is a binary number. If edge variable $e_{u,v}$ is assigned to true then the index of v is the successor of the index of u.

Example

Let $|V| = 7$, thus $k = \lceil \log_2 7 \rceil = 3$. For vertex v, variables v_2, v_4, and v_8 denote the least, middle, and most significant bit, respectively. For an edge variable $e_{u,v}$, we use the constraints:

$$\begin{aligned}
e_{u,v} &\rightarrow (u_2 \nleftrightarrow v_2) \\
(e_{u,v} \wedge \overline{u}_2) &\rightarrow (u_4 \leftrightarrow v_4) \\
(e_{u,v} \wedge u_2) &\rightarrow (u_4 \nleftrightarrow v_4) \\
(e_{u,v} \wedge \overline{u}_2) &\rightarrow (u_8 \leftrightarrow v_8) \\
(e_{u,v} \wedge \overline{u}_4) &\rightarrow (u_8 \leftrightarrow v_8) \\
(e_{u,v} \wedge u_2 \wedge u_4) &\rightarrow (u_8 \nleftrightarrow v_8)
\end{aligned}$$

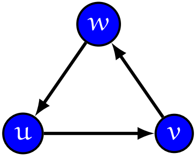
Hamiltonian Cycles: Binary Adder Encoding [Zhou 2020]

Each index is a binary number. If edge variable $e_{u,v}$ is assigned to true then the index of v is the successor of the index of u.

Example

Let $|V| = 7$, thus $k = \lceil \log_2 7 \rceil = 3$. For vertex v, variables v_2, v_4, and v_8 denote the least, middle, and most significant bit, respectively. For an edge variable $e_{u,v}$, we use the constraints:

$$
\begin{aligned}
e_{u,v} &\rightarrow (u_2 \nleftrightarrow v_2) \\
(e_{u,v} \wedge \overline{u}_2) &\rightarrow (u_4 \leftrightarrow v_4) \\
(e_{u,v} \wedge u_2) &\rightarrow (u_4 \nleftrightarrow v_4) \\
(e_{u,v} \wedge \overline{u}_2) &\rightarrow (u_8 \leftrightarrow v_8) \\
(e_{u,v} \wedge \overline{u}_4) &\rightarrow (u_8 \leftrightarrow v_8) \\
(e_{u,v} \wedge u_2 \wedge u_4) &\rightarrow (u_8 \nleftrightarrow v_8)
\end{aligned}
$$



$u_2 \rightarrow \neg v_2 \rightarrow w_2 \rightarrow \neg u_2$

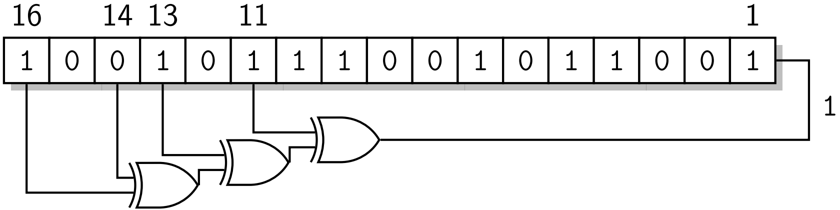
This encoding can quickly refute odd cycles

Hamiltonian Cycles: Linear-Feedback Shift Register

A k-bit Linear-Feedback Shift Register (LFSR) loops through $\{1, \ldots, 2^k - 1\}$ by shifting all bits one position to the left and placing the parity of some bits in the vacated position.

Example

An example LFSR of 16 bits is $x_{11} \oplus x_{13} \oplus x_{14} \oplus x_{16}$, which has $2^{16} - 1 = 65,535$ states. The figure below shows an illustration of this LFSR with state 1001011100101001. The next state is 0010111001010011.



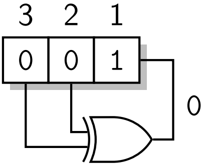
Hamiltonian Cycles: LFSR Encoding [Johnson 2018]

Enforcing the successor property using LFSR is compact and has been used to efficiently find Hamiltonian cycles in Erin and Stedman triples.

Example

Let $|V| = 7$, thus $k = \lceil \log_2(7+1) \rceil = 3$. We use 3-bit LFSR $x_2 \oplus x_3$. The bit-vector variables of vertex v are $v_{7,1}$, $v_{7,2}$, and $v_{7,3}$. For an edge variable $e_{u,v}$, we add the constraints:

$$
\begin{aligned}
e_{u,v} &\rightarrow (v_{7,1} \leftrightarrow (u_{7,2} \not\leftrightarrow u_{7,3})) \\
e_{u,v} &\rightarrow (v_{7,2} \leftrightarrow u_{7,1}) \\
e_{u,v} &\rightarrow (v_{7,3} \leftrightarrow u_{7,2})
\end{aligned}
$$



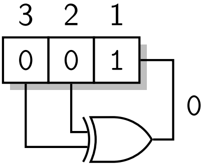
Hamiltonian Cycles: LFSR Encoding [Johnson 2018]

Enforcing the successor property using LFSR is compact and has been used to efficiently find Hamiltonian cycles in Erin and Stedman triples.

Example

Let $|V| = 7$, thus $k = \lceil \log_2(7+1) \rceil = 3$. We use 3-bit LFSR $x_2 \oplus x_3$. The bit-vector variables of vertex v are $v_{7,1}$, $v_{7,2}$, and $v_{7,3}$. For an edge variable $e_{u,v}$, we add the constraints:

$$
\begin{aligned}
e_{u,v} &\rightarrow (v_{7,1} \leftrightarrow (u_{7,2} \nleftrightarrow u_{7,3})) \\
e_{u,v} &\rightarrow (v_{7,2} \leftrightarrow u_{7,1}) \\
e_{u,v} &\rightarrow (v_{7,3} \leftrightarrow u_{7,2})
\end{aligned}
$$



This encoding is compact and has lots of propagation

Hamiltonian Cycles: Chinese Remainder Encoding [H '21]

Can we get the best all three worlds?

- Incremental SAT: Only partially encode the hard constraint
- Binary adder: refute some cycles quickly
- LFSR: few and short clauses, no auxiliary variables

Hamiltonian Cycles: Chinese Remainder Encoding [H '21]

Can we get the best all three worlds?

- Incremental SAT: Only partially encode the hard constraint
- Binary adder: refute some cycles quickly
- LFSR: few and short clauses, no auxiliary variables

Chinese remainder encoding:

- Block all subcycles except one of length $0 \pmod{m}$
- Pick m (can be smaller than $|V|$) with small prime factors
- Enforce $0 \pmod{p_i}$ for each prime factor p_i of m
- Use LFSR for primes > 2 and binary adder for $p_i = 2$

Hamiltonian Cycles: Flinders HCP Challenge Graphs

Evaluation on reasonably large instances from the Flinders HCP Challenge Graphs suite

- Runtime (s) of CaDiCaL on binary adder and LFSR
- Smallest k such that 2^k (or $2^k - 1$) is larger than $|V|$

graph #	$\lvert V \rvert$	$\lvert E \rvert$	adder (2^k)	LFSR ($2^k - 1$)
424	2466	4240	> 3600	> 3600
446	2557	4368	> 3600	> 3600
470	2740	4509	2500.61	> 3600
491	2844	4267	173.46	245.92
506	2964	4447	78.29	244.48
522	3060	4591	84.51	611.46
526	3108	4663	160.73	544.97
529	3132	4699	69.69	275.13

Hamiltonian Cycles: Chinese Remainder Results

Evaluation with CaDiCaL on various cycle lengths (m)

- ✗ : First solution consists of multiple cycles
- ✓ : First solution consists of a single cycle

graph #	2	6	12	60	105	420
424	9.81 ✗	665.18 ✗	340.11 ✗	307.71 ✗	494.11 ✓	**488.70** ✓
446	13.24 ✗	334.62 ✗	169.52 ✗	380.47 ✗	**573.38** ✓	722.23 ✓
470	17.08 ✗	166.16 ✗	152.31 ✗	933.36 ✗	501.91 ✗	**840.89** ✓
491	0.06 ✗	22.04 ✗	**7.47** ✓	34.45 ✓	123.36 ✓	135.22 ✓
506	0.11 ✗	31.75 ✗	**19.24** ✓	33.48 ✓	28.73 ✓	63.20 ✓
522	0.63 ✗	5.66 ✗	32.95 ✓	133.40 ✓	**30.40** ✓	67.03 ✓
526	0.05 ✗	24.16 ✗	71.67 ✓	**34.37** ✓	34.69 ✗	158.69 ✓
529	0.40 ✗	17.90 ✗	60.19 ✓	48.09 ✓	**42.33** ✓	365.58 ✓

Hamiltonian Cycles: Chinese Remainder Results

Evaluation with CaDiCaL on various cycle lengths (m)

- ✗ : First solution consists of multiple cycles
- ✓ : First solution consists of a single cycle

graph #	2	6	12	60	105	420
424	9.81 ✗	665.18 ✗	340.11 ✗	307.71 ✗	494.11 ✓	**488.70** ✓
446	13.24 ✗	334.62 ✗	169.52 ✗	380.47 ✗	**573.38** ✓	722.23 ✓
470	17.08 ✗	166.16 ✗	152.31 ✗	933.36 ✗	501.91 ✗	**840.89** ✓
491	0.06 ✗	22.04 ✗	**7.47** ✓	34.45 ✓	123.36 ✓	135.22 ✓
506	0.11 ✗	31.75 ✗	**19.24** ✓	33.48 ✓	28.73 ✓	63.20 ✓
522	0.63 ✗	5.66 ✗	32.95 ✓	133.40 ✓	**30.40** ✓	67.03 ✓
526	0.05 ✗	24.16 ✗	71.67 ✓	**34.37** ✓	34.69 ✗	158.69 ✓
529	0.40 ✗	17.90 ✗	60.19 ✓	48.09 ✓	**42.33** ✓	365.58 ✓

Trusting a no Ham. cycle result requires verifying the encoding

Further reading

More details about cardinality encodings can be found in:

- Sinz's encoding:
  Carsten Sinz. Towards an Optimal CNF Encoding of Boolean Cardinality Constraints. CP 2005. pp. 827-831
  `http://www.carstensinz.de/papers/CP-2005.pdf`
- Totalizer encoding:
  Olivier Bailleux, Yacine Boufkhad. Efficient CNF Encoding of Boolean Cardinality Constraints. CP 2003. pp. 108-122
  `https://tinyurl.com/y6ph76au`
- Modulo Totalizer encoding:
  Toru Ogawa, Yangyang Liu, Ryuzo Hasegawa, Miyuki Koshimura, Hiroshi Fujita. Modulo Based CNF Encoding of Cardinality Constraints and Its Application to MaxSAT Solvers. ICTAI 2013. pp. 9-17 `https://ieeexplore.ieee.org/document/6735224`
- Cardinality networks:
  Roberto Asin, Robert Nieuwenhuis, Albert Oliveras, Enric Rodriguez-Carbonell. Cardinality Networks and Their Applications. SAT 2009. pp. 167-180 `https://tinyurl.com/yxwrxzxo`