Symbolic AI, automated reasoning and neuro-symbolic AI

Can we learn *and* guarantee?

Thomas Schiex

INRAE, MIAT — ANITI, Toulouse

IA2 2026 research school — Monday, 2–4 p.m.

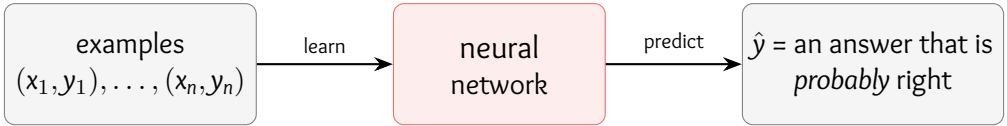
RÉPUBLIQUE FRANÇAISE
Liberté Égalité Fraternité

INRAE

ANITI

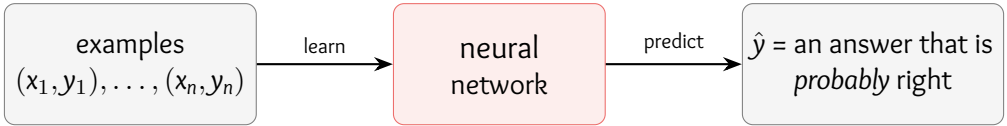
Université de Toulouse

A neural network predicts… with no guarantee



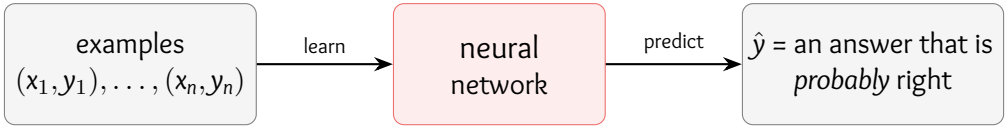
- From the previous lecture: deep learning (DL) generalises from examples
- No guarantee on any individual answer: an obvious rule can be violated
- No explicit knowledge to inspect or to correct

A neural network predicts… with no guarantee



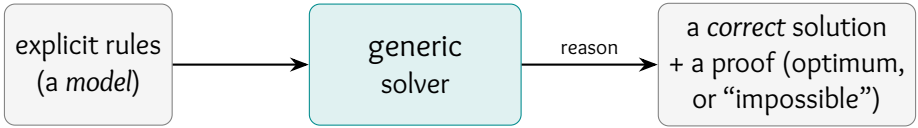
- From the previous lecture: deep learning (DL) generalises from examples
- No guarantee on any individual answer: an obvious rule can be violated
- No explicit knowledge to inspect or to correct

A neural network predicts… with no guarantee



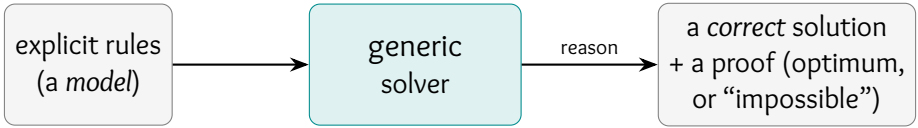
- From the previous lecture: deep learning (DL) generalises from examples
- No guarantee on any individual answer: an obvious rule can be violated
- No explicit knowledge to inspect or to correct

A solver guarantees... if you give it the rules



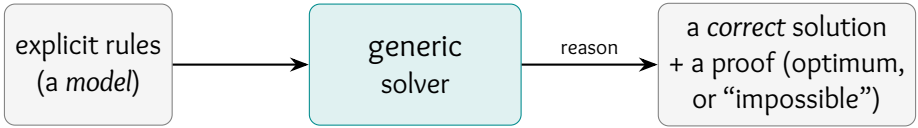
- Automated reasoning derives *certain* consequences
- The price: you must write the knowledge down, and the computation can be long
- What if the rules are unknown or fuzzy (biology, chemistry, medicine)?

A solver guarantees... if you give it the rules



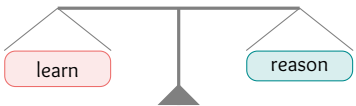
- Automated reasoning derives *certain* consequences
- The price: you must write the knowledge down, and the computation can be long
- What if the rules are unknown or fuzzy (biology, chemistry, medicine)?

A solver guarantees… if you give it the rules



- Automated reasoning derives *certain* consequences
- The price: you must write the knowledge down, and the computation can be long
- What if the rules are unknown or fuzzy (biology, chemistry, medicine)?

Can we learn *and* guarantee?



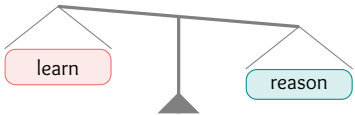
A The formalisms of reasoning
B A little complexity
C How solvers reason + demo
D Neuro-symbolic: definitions

— break —

E Case study: protein design
F Learning the rules: Effie
G Proofs and certificates
H Limits and take-home messages

Landmark: the dot at the bottom of each slide shows the current part.

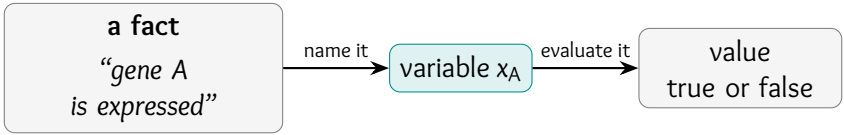
A. From symbols to a function, and questions about it



Can we learn and guarantee?

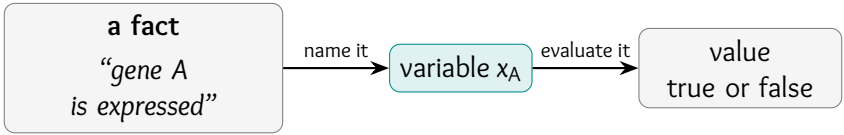
- A symbol, a fact, a variable
- Facts interact: a global function
- Reasoning means querying it, with guarantees
- A language to describe it: SAT, CP, costs, probabilities

A symbol denotes a fact, a variable carries its value



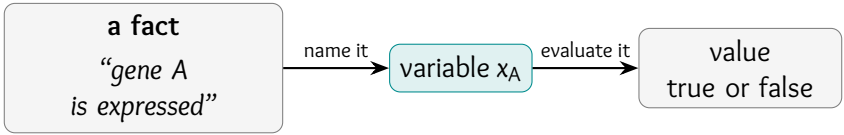
- The symbol is *arbitrary*; what matters is the fact it denotes
- Elsewhere: “cell $(1, 1)$ holds a 5”, “reaction r takes place”, “position 17 carries a leucine”
- One fact = one variable: this is the step from the world to the model

A symbol denotes a fact, a variable carries its value



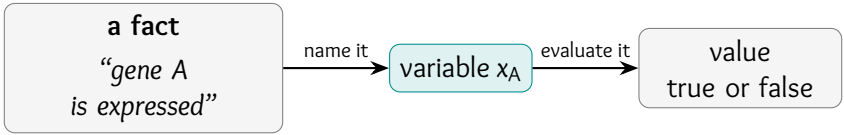
- The symbol is *arbitrary*; what matters is the fact it denotes
- Elsewhere: “cell $(1, 1)$ holds a 5”, “reaction r takes place”, “position 17 carries a leucine”
- One fact = one variable: this is the step from the world to the model

A symbol denotes a fact, a variable carries its value



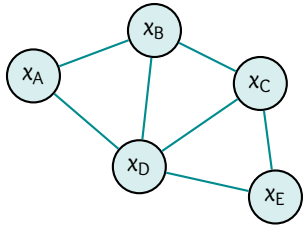
- The symbol is *arbitrary*; what matters is the fact it denotes
- Elsewhere: “cell $(1, 1)$ holds a 5”, “reaction r takes place”, “position 17 carries a leucine”
- One fact = one variable: this is the step from the world to the model

A symbol denotes a fact, a variable carries its value



- The symbol is *arbitrary*; what matters is the fact it denotes
- Elsewhere: “cell $(1, 1)$ holds a 5”, “reaction r takes place”, “position 17 carries a leucine”
- One fact = one variable: this is the step from the world to the model

Many interacting facts: a global property

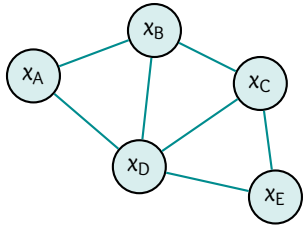


global property
"the pathway produces metabolite M"

$f(x_A, \ldots, x_E) \in \{\text{true}, \text{false}\}$

- n facts $\Rightarrow 2^n$ possible states of the system
- The global property *is* a function: to each state it assigns true or false
- Everything reads off it: $f = g$, the formulas are *equivalent* ($f \Leftrightarrow g$); $f \leq g$ everywhere, f *entails* g ($f \models g$)

Many interacting facts: a global property

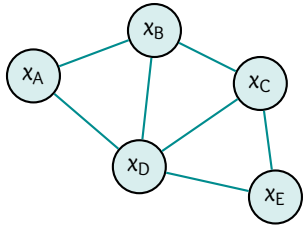


global property
"the pathway produces metabolite M"

$f(x_A, \ldots, x_E) \in \{\text{true}, \text{false}\}$

- n facts $\Rightarrow 2^n$ possible states of the system
- The global property *is* a function: to each state it assigns true or false
- Everything reads off it: $f = g$, the formulas are *equivalent* ($f \Leftrightarrow g$); $f \leq g$ everywhere, f *entails* g ($f \models g$)

Many interacting facts: a global property

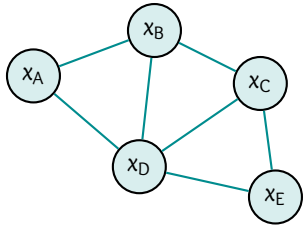


global property
"the pathway produces metabolite M"

$f(x_A, \ldots, x_E) \in \{\text{true}, \text{false}\}$

- n facts $\Rightarrow 2^n$ possible states of the system
- The global property *is* a function: to each state it assigns true or false
- Everything reads off it: $f = g$, the formulas are *equivalent* ($f \Leftrightarrow g$); $f \leq g$ everywhere, f *entails* g ($f \models g$)

Many interacting facts: a global property

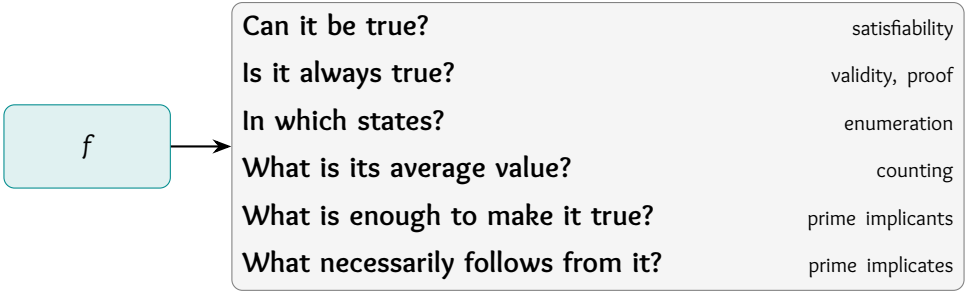


global property
"the pathway produces metabolite M"

$f(x_A, \ldots, x_E) \in \{\text{true}, \text{false}\}$

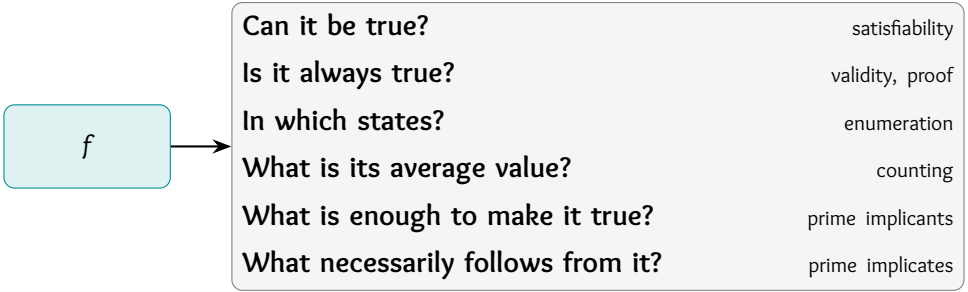
- n facts $\Rightarrow 2^n$ possible states of the system
- The global property *is* a function: to each state it assigns true or false
- Everything reads off it: $f = g$, the formulas are *equivalent* ($f \Leftrightarrow g$); $f \leq g$ everywhere, f *entails* g ($f \models g$)

Reasoning means querying this function — with guarantees



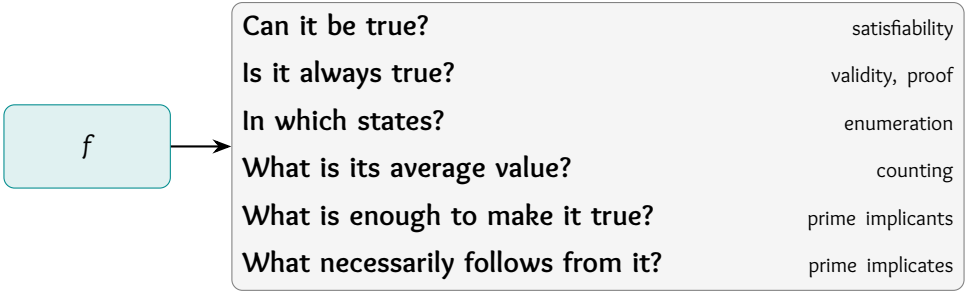
- Automated reasoning answers **with guarantees**: the answer is proved, not estimated
- "No" is as useful an answer as "yes" — and much harder to obtain
- One question remains: how do we *define* this function? And if we cannot: *learn* it (part F)

Reasoning means querying this function — with guarantees



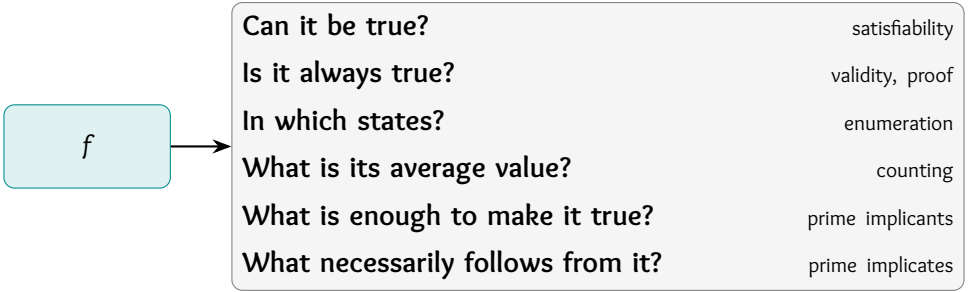
- Automated reasoning answers **with guarantees**: the answer is proved, not estimated
- "No" is as useful an answer as "yes" — and much harder to obtain
- One question remains: how do we *define* this function? And if we cannot: *learn* it (part F)

Reasoning means querying this function — with guarantees



- Automated reasoning answers **with guarantees**: the answer is proved, not estimated
- "No" is as useful an answer as "yes" — and much harder to obtain
- One question remains: how do we *define* this function? And if we cannot: *learn* it (part F)

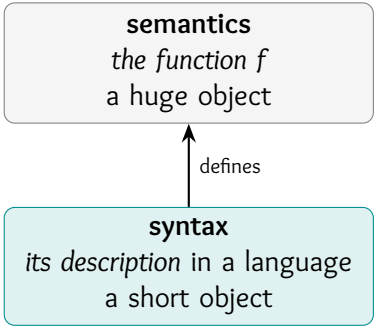
Reasoning means querying this function — with guarantees



- Automated reasoning answers **with guarantees**: the answer is proved, not estimated
- "No" is as useful an answer as "yes" — and much harder to obtain
- One question remains: how do we *define* this function? And if we cannot: *learn* it (part F)

We never write the function out in full

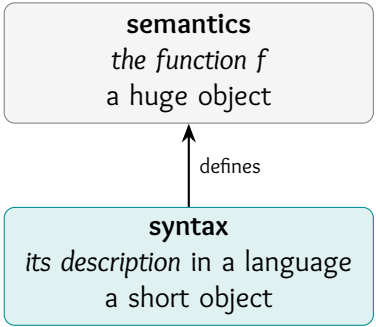
- Truth table: 2^n rows
- $n = 300$ facts $\Rightarrow 2^{300} \approx 10^{90}$ states, more than there are atoms in the observable universe
- So we do not enumerate it: we *describe* it



- The whole game: a language **expressive** enough to say what we mean, **constrained** enough that we can reason about it

We never write the function out in full

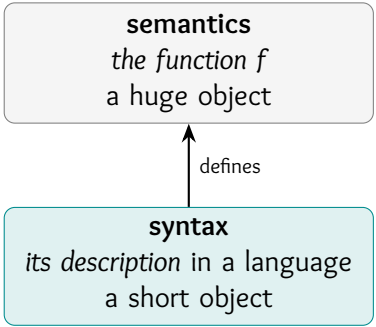
- Truth table: 2^n rows
- $n = 300$ facts $\Rightarrow 2^{300} \approx 10^{90}$ states, more than there are atoms in the observable universe
- So we do not enumerate it: we *describe* it



- The whole game: a language **expressive** enough to say what we mean, **constrained** enough that we can reason about it

We never write the function out in full

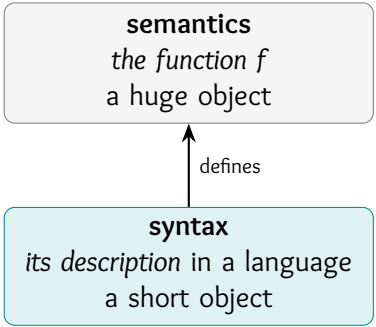
- Truth table: 2^n rows
- $n = 300$ facts $\Rightarrow 2^{300} \approx 10^{90}$ states, more than there are atoms in the observable universe
- So we do not enumerate it: we *describe* it



- The whole game: a language **expressive** enough to say what we mean, **constrained** enough that we can reason about it

We never write the function out in full

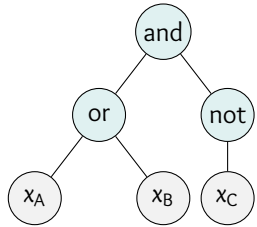
- Truth table: 2^n rows
- $n = 300$ facts $\Rightarrow 2^{300} \approx 10^{90}$ states, more than there are atoms in the observable universe
- So we do not enumerate it: we *describe* it



- The whole game: a language **expressive** enough to say what we mean, **constrained** enough that we can reason about it

A compositional language: propositional logic

- Three connectives: and, or, not
- **Compositional**: the meaning of the whole is computed from the meaning of the parts
- A *short* formula can describe a function over 2^n states



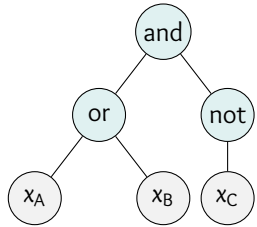
$$(x_A \vee x_B) \wedge \neg x_C$$

"the pathway works if A or B is expressed, and C is not"

(Biere et al. 2021)

A compositional language: propositional logic

- Three connectives: and, or, not
- **Compositional**: the meaning of the whole is computed from the meaning of the parts
- A *short* formula can describe a function over 2^n states



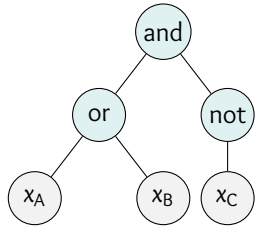
$$(x_A \vee x_B) \wedge \neg x_C$$

"the pathway works if A or B is expressed, and C is not"

(Biere et al. 2021)

A compositional language: propositional logic

- Three connectives: and, or, not
- **Compositional**: the meaning of the whole is computed from the meaning of the parts
- A *short* formula can describe a function over 2^n states



$$(x_A \vee x_B) \wedge \neg x_C$$

"the pathway works if A or B is expressed, and C is not"

(Biere et al. 2021)

SAT: clauses, and a single question

- Conjunctive normal form: an **and** of **clauses**, each clause an **or** of literals
- Every Boolean function can be written this way
- The SAT question: **is the maximum of f equal to 1?**
  is there a state that makes it true?
- Yes $\Rightarrow$ the state is returned; no $\Rightarrow$ it is *proved*

$$(x_A \vee x_B) \wedge (\neg x_C) \wedge (\neg x_A \vee x_D)$$

CDCL solvers: Kissat, CaDiCaL

millions of variables, routinely

(Biere et al. 2021)

SAT: clauses, and a single question

- Conjunctive normal form: an **and** of **clauses**, each clause an **or** of literals
- Every Boolean function can be written this way
- The SAT question: **is the maximum of f equal to 1?**
  is there a state that makes it true?
- Yes $\Rightarrow$ the state is returned; no $\Rightarrow$ it is *proved*

$$(x_A \vee x_B) \wedge (\neg x_C) \wedge (\neg x_A \vee x_D)$$

CDCL solvers: Kissat, CaDiCaL

millions of variables, routinely

(Biere et al. 2021)

SAT: clauses, and a single question

- Conjunctive normal form: an **and** of **clauses**, each clause an **or** of literals
- Every Boolean function can be written this way
- The SAT question: **is the maximum of f equal to 1?**
  is there a state that makes it true?
- Yes ⇒ the state is returned; no ⇒ it is *proved*

$$(x_A \vee x_B) \wedge (\neg x_C) \wedge (\neg x_A \vee x_D)$$

CDCL solvers: Kissat, CaDiCaL

millions of variables, routinely

(Biere et al. 2021)

SAT: clauses, and a single question

- Conjunctive normal form: an **and** of **clauses**, each clause an **or** of literals
- Every Boolean function can be written this way
- The SAT question: **is the maximum of f equal to 1?**
  is there a state that makes it true?
- Yes $\Rightarrow$ the state is returned; no $\Rightarrow$ it is *proved*

$$(x_A \vee x_B) \wedge (\neg x_C) \wedge (\neg x_A \vee x_D)$$

CDCL solvers: Kissat, CaDiCaL

millions of variables, routinely

(Biere et al. 2021)

First slider: the *domain* of the variables

Sudoku in SAT

- 9 Booleans per cell: $x_{r,c,v}$
- "at least one value": one clause of 9 literals
- "at most one value": 36 clauses per cell
- 729 variables, thousands of clauses

Sudoku in CP

- **one** variable per cell, domain $\{1, \ldots, 9\}$
- "exactly one value" is free: that is the domain
- language: **tables** and **global constraints** (`AllDifferent`)
- 81 variables, 27 constraints

- Same function, far more compact description — and far stronger propagation

(Rossi et al. 2006)

First slider: the *domain* of the variables

Sudoku in SAT

- 9 Booleans per cell: $x_{r,c,v}$
- "at least one value": one clause of 9 literals
- "at most one value": 36 clauses per cell
- 729 variables, thousands of clauses

Sudoku in CP

- **one** variable per cell, domain $\{1, \ldots, 9\}$
- "exactly one value" is free: that is the domain
- language: **tables** and **global constraints** (`AllDifferent`)
- 81 variables, 27 constraints

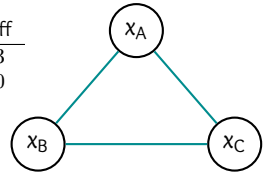
- Same function, far more compact description — and far stronger propagation

(Rossi et al. 2006)

Second slider: the *co-domain* of the function

- Replace $\{\text{true}, \text{false}\}$ with $\mathbb{N} \cup \{\infty\}$: **costs**
- Combination becomes a **sum** instead of an **and**
- The query becomes a **minimum** instead of a maximum
- Cost ∞ = forbidden: constraints are a special case

x_A \ x_B	on	off
on	0	3
off	∞	0



$$f(x) = \sum_{\text{tables}} \quad \text{then } \min_x f(x)$$

"A off and B on: impossible;

A on and B off: costly but possible"

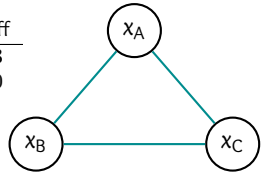
Cost Function Networks (CFN) — solver `toulbar2`

(Cooper et al. 2020), (Hurley et al. 2016)

Second slider: the *co-domain* of the function

- Replace $\{\text{true}, \text{false}\}$ with $\mathbb{N} \cup \{\infty\}$: **costs**
- Combination becomes a **sum** instead of an **and**
- The query becomes a **minimum** instead of a maximum
- Cost ∞ = forbidden: constraints are a special case

x_A \ x_B	on	off
on	0	3
off	∞	0



$$f(x) = \sum_{\text{tables}} \quad \text{then } \min_x f(x)$$

"A off and B on: impossible;

A on and B off: costly but possible"

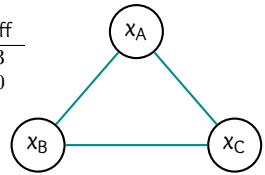
Cost Function Networks (CFN) — solver `toulbar2`

(Cooper et al. 2020), (Hurley et al. 2016)

Second slider: the *co-domain* of the function

- Replace $\{\text{true}, \text{false}\}$ with $\mathbb{N} \cup \{\infty\}$: **costs**
- Combination becomes a **sum** instead of an **and**
- The query becomes a **minimum** instead of a maximum
- Cost ∞ = forbidden: constraints are a special case

x_A \ x_B	on	off
on	0	3
off	∞	0



$$f(x) = \sum_{\text{tables}} \quad \text{then } \min_x f(x)$$

"A off and B on: impossible;

A on and B off: costly but possible"

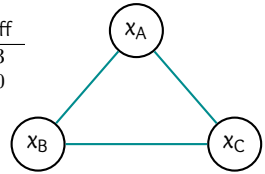
Cost Function Networks (CFN) — solver `toulbar2`

(Cooper et al. 2020), (Hurley et al. 2016)

Second slider: the *co-domain* of the function

- Replace $\{\text{true}, \text{false}\}$ with $\mathbb{N} \cup \{\infty\}$: **costs**
- Combination becomes a **sum** instead of an **and**
- The query becomes a **minimum** instead of a maximum
- Cost ∞ = forbidden: constraints are a special case

x_A \ x_B	on	off
on	0	3
off	∞	0



$$f(x) = \sum_{\text{tables}} \quad \text{then } \min_x f(x)$$

"A off and B on: impossible;

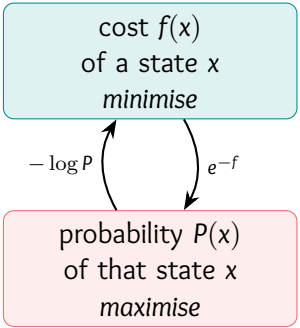
A on and B off: costly but possible"

Cost Function Networks (CFN) — solver `toulbar2`

(Cooper et al. 2020), (Hurley et al. 2016)

Third slider: the probabilistic reading

- Boltzmann: $P(x) \propto \exp(-f(x))$
  a small cost = a high probability
- We obtain a **Markov random field** (MRF)
- Normalised tables + acyclic graph ⇒ **Bayesian network**
- A new query: $Z = \sum_x \exp(-f(x))$, the normalising constant

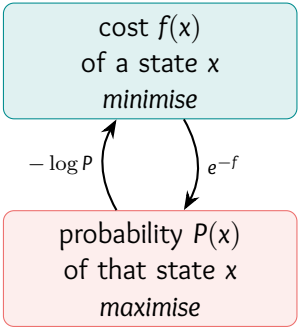


one object, two readings:
the most probable state
= the minimal cost (MAP)

(Cooper et al. 2020), (Koller et al. 2009)

Third slider: the probabilistic reading

- Boltzmann: $P(x) \propto \exp(-f(x))$
  a small cost = a high probability
- We obtain a **Markov random field** (MRF)
- Normalised tables + acyclic graph ⇒ **Bayesian network**
- A new query: $Z = \sum_x \exp(-f(x))$, the normalising constant

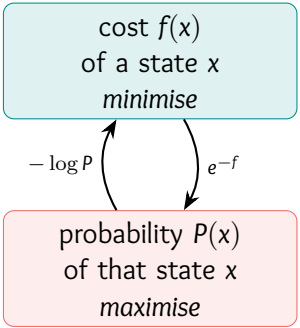


one object, two readings:
the most probable state
= the minimal cost (MAP)

(Cooper et al. 2020), (Koller et al. 2009)

Third slider: the probabilistic reading

- Boltzmann: $P(x) \propto \exp(-f(x))$
  a small cost = a high probability
- We obtain a **Markov random field** (MRF)
- Normalised tables + acyclic graph ⇒ **Bayesian network**
- A new query: $Z = \sum_x \exp(-f(x))$, the normalising constant

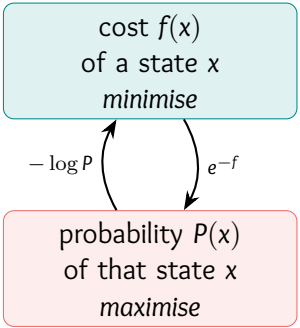


one object, two readings:
the most probable state
= the minimal cost (MAP)

(Cooper et al. 2020), (Koller et al. 2009)

Third slider: the probabilistic reading

- Boltzmann: $P(x) \propto \exp(-f(x))$
  a small cost = a high probability
- We obtain a **Markov random field** (MRF)
- Normalised tables + acyclic graph ⇒ **Bayesian network**
- A new query: $Z = \sum_x \exp(-f(x))$, the normalising constant



one object, two readings:
the most probable state
= the minimal cost (MAP)

(Cooper et al. 2020), (Koller et al. 2009)

The same function, a few choices

	Domain	Co-domain	Comb.	Typical query
SAT	$\{0,1\}$	$\{0,1\}$	$\wedge$	$\max f = 1$?
Constraints (CP)	finite	$\{0,1\}$	$\wedge$	$\max f = 1$?
Costs (CFN)	finite	$\mathbb{N} \cup \{\infty\}$	$+$	$\min f$
Possibilistic	finite	$[0,1]$ ordinal	$\min$	$\max f$ (max-min)
Markov random field	finite	$\mathbb{R}^+$	$\times$	$\max P$ (MAP), Z
Bayesian network	finite	$[0,1]$	$\times$	$P(\cdot \mid \cdot)$, Z

- The *language* varies too: clauses, tables, global constraints and global cost functions
- One common framework: **discrete graphical models** — on the constraint side, **VCSPs** (valued constraint satisfaction problems)

(Cooper et al. 2020), (Dubois et al. 1994), (Schiex et al. 1995)

The same function, a few choices

	Domain	Co-domain	Comb.	Typical query
SAT	$\{0,1\}$	$\{0,1\}$	$\wedge$	$\max f = 1$?
Constraints (CP)	finite	$\{0,1\}$	$\wedge$	$\max f = 1$?
Costs (CFN)	finite	$\mathbb{N} \cup \{\infty\}$	$+$	$\min f$
Possibilistic	finite	$[0,1]$ ordinal	$\min$	$\max f$ (max-min)
Markov random field	finite	$\mathbb{R}^+$	$\times$	$\max P$ (MAP), Z
Bayesian network	finite	$[0,1]$	$\times$	$P(\cdot \mid \cdot)$, Z

- The *language* varies too: clauses, tables, global constraints and global cost functions
- One common framework: **discrete graphical models** — on the constraint side, **VCSPs** (valued constraint satisfaction problems)

(Cooper et al. 2020), (Dubois et al. 1994), (Schiex et al. 1995)

The same function, a few choices

	Domain	Co-domain	Comb.	Typical query
SAT	$\{0,1\}$	$\{0,1\}$	$\wedge$	$\max f = 1$?
Constraints (CP)	finite	$\{0,1\}$	$\wedge$	$\max f = 1$?
Costs (CFN)	finite	$\mathbb{N} \cup \{\infty\}$	$+$	$\min f$
Possibilistic	finite	$[0,1]$ ordinal	$\min$	$\max f$ (max-min)
Markov random field	finite	$\mathbb{R}^+$	$\times$	$\max P$ (MAP), Z
Bayesian network	finite	$[0,1]$	$\times$	$P(\cdot \mid \cdot)$, Z

- The *language* varies too: clauses, tables, global constraints and global cost functions
- One common framework: **discrete graphical models** — on the constraint side, **VCSPs** (valued constraint satisfaction problems)

(Cooper et al. 2020), (Dubois et al. 1994), (Schiex et al. 1995)

Other languages for other needs

SMT: logic + theories

- Integers, reals, arrays, bit-vectors
- Z3, cvc5; code verification

```
x, y = Ints('x y')
s.add(x + 2*y == 7, x > y, y > 0)
s.check()      # sat: y=1, x=5
```

ASP: rules and exceptions

- Default negation (exceptions)
- clingo; configuration, biology

```
flies(X) :- bird(X),
            not penguin(X).
bird(tweety).  bird(pingu).
penguin(pingu).
```

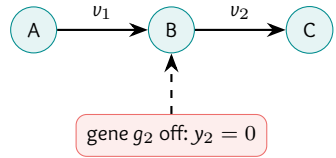
Answer: flies(tweety), but **not** flies(pingu)

The same scheme everywhere: a syntax, a semantics, queries, dedicated solvers.

(Moura et al. 2008), (Barbosa et al. 2022), (Gebser et al. 2019)

When variables become continuous: MILP

- Integer *and* real variables, **linear** constraints and objective
- Tools: CPLEX, Gurobi, HiGHS, SCIP
- Example: metabolic fluxes at steady state (FBA) + Boolean gene regulation rules
- The 0/1 "gene active" variables switch fluxes off ⇒ a MILP

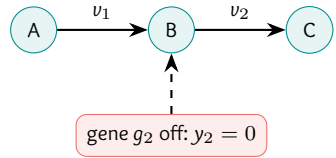


$$v_1 = v_2 \quad \text{(steady state)}$$
$$0 \le v_2 \le M y_2,\ y_2 \in \{0, 1\}$$
$$\max\ v_{\text{biomass}}$$

(Shlomi et al. 2007)

When variables become continuous: MILP

- Integer *and* real variables, **linear** constraints and objective
- Tools: CPLEX, Gurobi, HiGHS, SCIP
- Example: metabolic fluxes at steady state (FBA) + Boolean gene regulation rules
- The 0/1 "gene active" variables switch fluxes off ⇒ a MILP

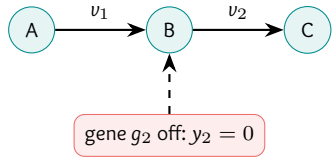


$$v_1 = v_2 \quad \text{(steady state)}$$
$$0 \le v_2 \le M y_2,\ y_2 \in \{0, 1\}$$
$$\max\ v_{\text{biomass}}$$

(Shlomi et al. 2007)

When variables become continuous: MILP

- Integer *and* real variables, **linear** constraints and objective
- Tools: CPLEX, Gurobi, HiGHS, SCIP
- Example: metabolic fluxes at steady state (FBA) + Boolean gene regulation rules
- The 0/1 "gene active" variables switch fluxes off $\Rightarrow$ a MILP



$$v_1 = v_2 \quad \text{(steady state)}$$
$$0 \leq v_2 \leq M y_2,\ y_2 \in \{0, 1\}$$
$$\max\ v_{\text{biomass}}$$

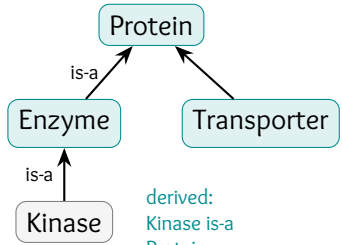
(Shlomi et al. 2007)

When variables are no longer enough: objects and relations

- First-order logic: *objects*, *relations*, quantifiers

  $\forall x\,(\text{Enzyme}(x) \Rightarrow \text{Protein}(x))$

- Very expressive… and undecidable (part B)
- Description logics: decidable fragments, for *ontologies* (OWL, Gene Ontology, SNOMED CT)
- **Tomorrow**: knowledge graphs

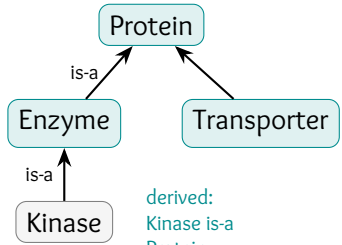


(Baader et al. 2017)

When variables are no longer enough: objects and relations

- First-order logic: *objects*, *relations*, quantifiers

  $\forall x\,(\text{Enzyme}(x) \Rightarrow \text{Protein}(x))$
- Very expressive... and undecidable (part B)
- Description logics: decidable fragments, for *ontologies* (OWL, Gene Ontology, SNOMED CT)
- **Tomorrow**: knowledge graphs

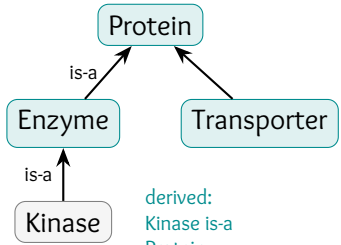


(Baader et al. 2017)

When variables are no longer enough: objects and relations

- First-order logic: *objects*, *relations*, quantifiers

  $\forall x\,(\text{Enzyme}(x) \Rightarrow \text{Protein}(x))$
- Very expressive… and undecidable (part B)
- Description logics: decidable fragments, for *ontologies* (OWL, Gene Ontology, SNOMED CT)
- **Tomorrow**: knowledge graphs

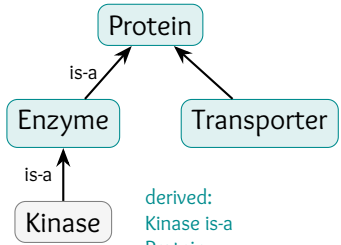


(Baader et al. 2017)

When variables are no longer enough: objects and relations

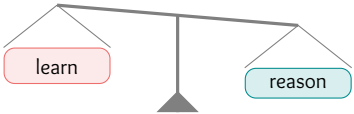
- First-order logic: *objects*, *relations*, quantifiers

  $\forall x\,(\text{Enzyme}(x) \Rightarrow \text{Protein}(x))$
- Very expressive… and undecidable (part B)
- Description logics: decidable fragments, for *ontologies* (OWL, Gene Ontology, SNOMED CT)
- **Tomorrow**: knowledge graphs



(Baader et al. 2017)

B. A little complexity

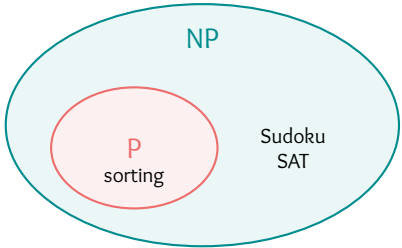


Can we learn and guarantee?

- The queries of part A are hard
- Checking versus finding; SAT
- Optimising, counting, and beyond
- Worst case versus practice

Checking a solution is easy, finding one can be hard

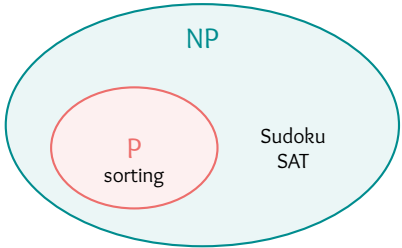
- **P**: problems *solved* in polynomial time (*e.g.* sorting, shortest path)
- **NP**: problems whose candidate solution can be *checked* in polynomial time
- Sudoku: checking a filled grid = 27 tests; filling it in…
- P = NP? Still open (a Clay Institute prize)



(Arora et al. 2009)

Checking a solution is easy, finding one can be hard

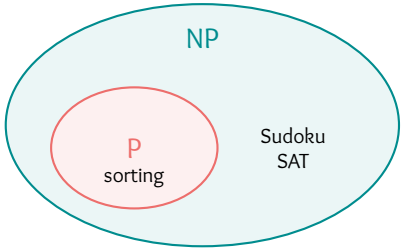
- **P**: problems *solved* in polynomial time (*e.g.* sorting, shortest path)
- **NP**: problems whose candidate solution can be *checked* in polynomial time
- Sudoku: checking a filled grid = 27 tests; filling it in…
- P = NP? Still open (a Clay Institute prize)



(Arora et al. 2009)

Checking a solution is easy, finding one can be hard

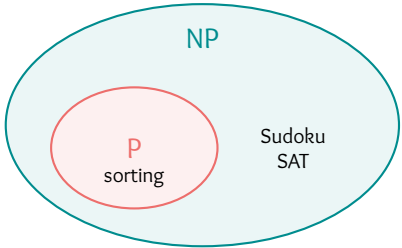
- **P**: problems *solved* in polynomial time (*e.g.* sorting, shortest path)
- **NP**: problems whose candidate solution can be *checked* in polynomial time
- Sudoku: checking a filled grid = 27 tests; filling it in…
- P = NP? Still open (a Clay Institute prize)



(Arora et al. 2009)

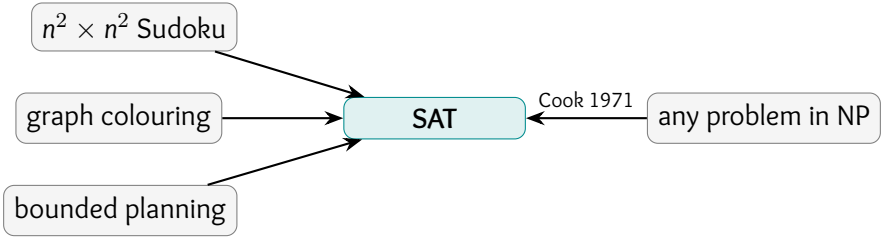
Checking a solution is easy, finding one can be hard

- **P**: problems *solved* in polynomial time (*e.g.* sorting, shortest path)
- **NP**: problems whose candidate solution can be *checked* in polynomial time
- Sudoku: checking a filled grid = 27 tests; filling it in…
- P = NP? Still open (a Clay Institute prize)



(Arora et al. 2009)

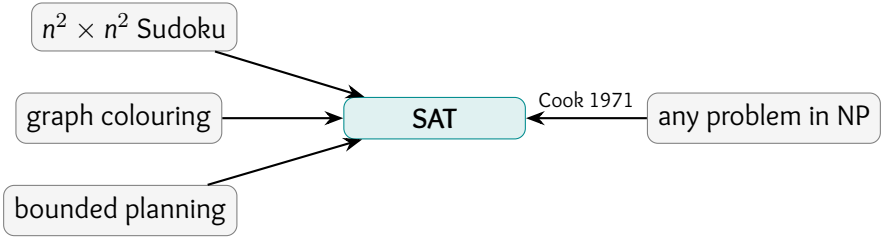
SAT is the first NP-complete problem



- NP-complete: in NP, and every problem in NP *translates* into it efficiently
- So a good SAT solver is a solver for all of NP
- This is why so many problems can be "compiled" into SAT, CP, MILP…

(Cook 1971), (Yato et al. 2003)

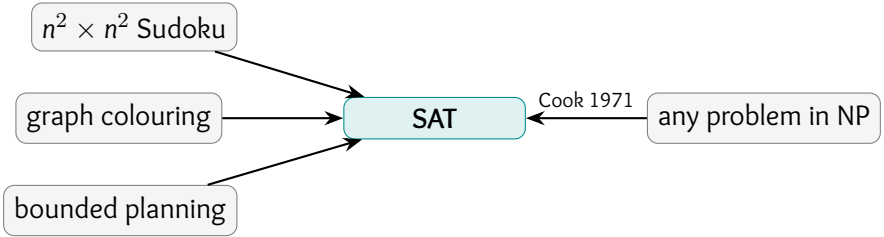
SAT is the first NP-complete problem



- NP-complete: in NP, and every problem in NP *translates* into it efficiently
- So a good SAT solver is a solver for all of NP
- This is why so many problems can be "compiled" into SAT, CP, MILP…

(Cook 1971), (Yato et al. 2003)

SAT is the first NP-complete problem

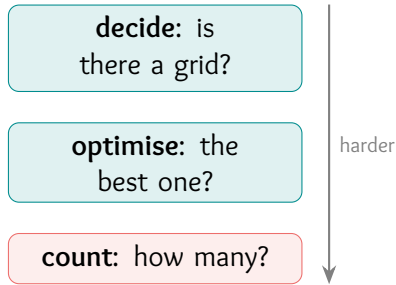


- NP-complete: in NP, and every problem in NP *translates* into it efficiently
- So a good SAT solver is a solver for all of NP
- This is why so many problems can be "compiled" into SAT, CP, MILP...

(Cook 1971), (Yato et al. 2003)

Optimising and counting: at least as hard as deciding

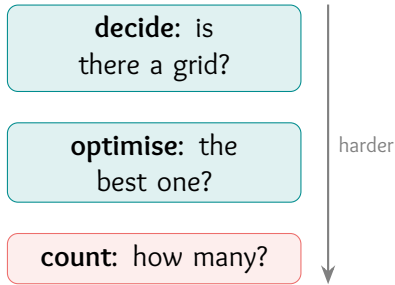
- **NP-hard:** at least as hard as anything in NP (*e.g.* the $\min f$ of a CFN, the MAP of a Markov random field)
- #P: *counting* the solutions (*e.g.* the normalising constant Z seen in part A)
- For learning: the exact likelihood of a CFN requires Z
  $\Rightarrow$ a #P-hard problem (part F)



(Arora et al. 2009)

Optimising and counting: at least as hard as deciding

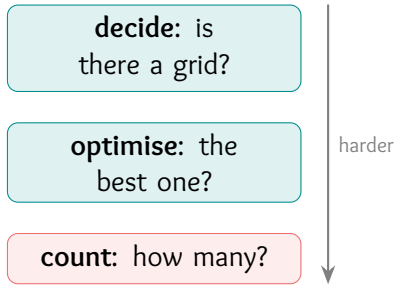
- **NP-hard:** at least as hard as anything in NP (*e.g.* the $\min f$ of a CFN, the MAP of a Markov random field)
- **#P:** *counting* the solutions (*e.g.* the normalising constant Z seen in part A)
- For learning: the exact likelihood of a CFN requires Z
  $\Rightarrow$ a #P-hard problem (part F)



(Arora et al. 2009)

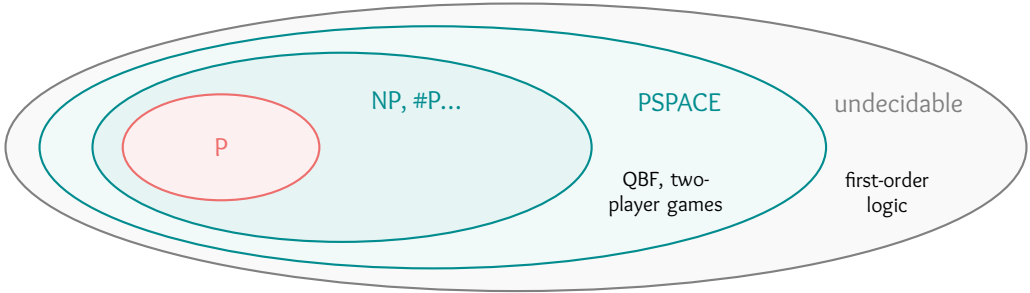
Optimising and counting: at least as hard as deciding

- **NP-hard:** at least as hard as anything in NP (*e.g.* the $\min f$ of a CFN, the MAP of a Markov random field)
- **#P:** *counting* the solutions (*e.g.* the normalising constant Z seen in part A)
- For learning: the exact likelihood of a CFN requires Z
  $\Rightarrow$ a #P-hard problem (part F)



(Arora et al. 2009)

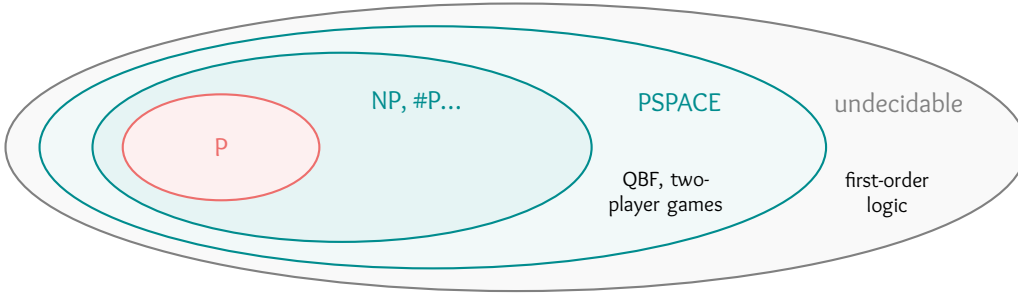
Beyond: PSPACE, and even the undecidable



- Quantified formulas (QBF): “for every x, there is a *y*...” — two-player games
- First-order logic: a prover may never halt
- Hence the appeal of description logics: decidable (and sometimes polynomial)

(Baader et al. 2017)

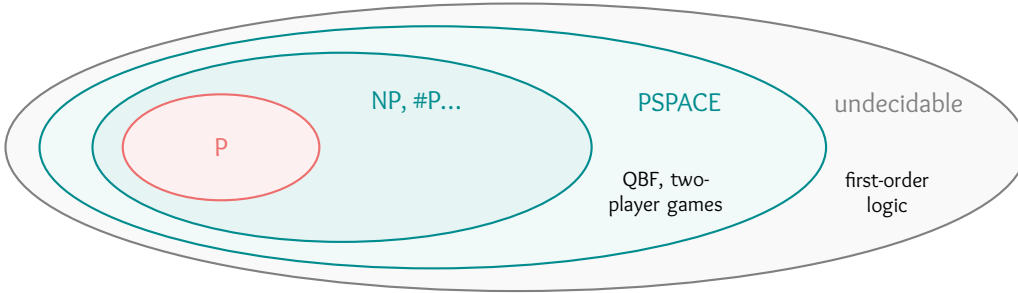
Beyond: PSPACE, and even the undecidable



- Quantified formulas (QBF): “for every x, there is a *y*…” — two-player games
- First-order logic: a prover may never halt
- Hence the appeal of description logics: decidable (and sometimes polynomial)

(Baader et al. 2017)

Beyond: PSPACE, and even the undecidable

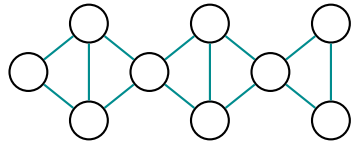


- Quantified formulas (QBF): “for every x, there is a *y*…” — two-player games
- First-order logic: a prover may never halt
- Hence the appeal of description logics: decidable (and sometimes polynomial)

(Baader et al. 2017)

Worst case $\neq$ practice: structure saves you

- NP-hard = hard *in the worst case*, not on your instances
- Polynomial fragments: 2-SAT, Horn clauses, minimum cut (submodular functions)
- **Treewidth** w: if the graph of the problem looks like a tree, cost d^{w+1} instead of d^n
- Modern solvers: learning, heuristics, restarts

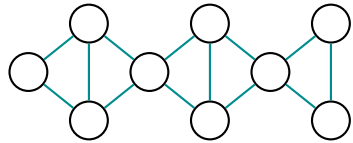


a chain of small clusters:
treewidth 2, easy

(Cooper et al. 2020)

Worst case $\neq$ practice: structure saves you

- NP-hard = hard *in the worst case*, not on your instances
- Polynomial fragments: 2-SAT, Horn clauses, minimum cut (submodular functions)
- **Treewidth** w: if the graph of the problem looks like a tree, cost d^{w+1} instead of d^n
- Modern solvers: learning, heuristics, restarts

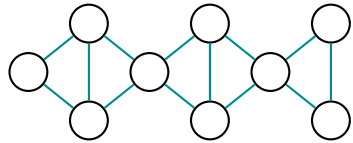


a chain of small clusters:
treewidth 2, easy

(Cooper et al. 2020)

Worst case $\neq$ practice: structure saves you

- NP-hard = hard *in the worst case*, not on your instances
- Polynomial fragments: 2-SAT, Horn clauses, minimum cut (submodular functions)
- **Treewidth** w: if the graph of the problem looks like a tree, cost d^{w+1} instead of d^n
- Modern solvers: learning, heuristics, restarts

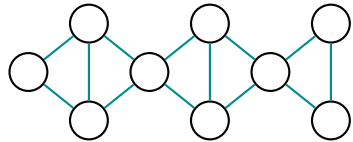


a chain of small clusters:
treewidth 2, easy

(Cooper et al. 2020)

Worst case $\neq$ practice: structure saves you

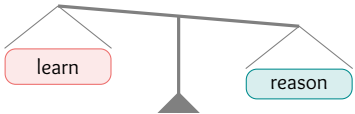
- NP-hard = hard *in the worst case*, not on your instances
- Polynomial fragments: 2-SAT, Horn clauses, minimum cut (submodular functions)
- **Treewidth** w: if the graph of the problem looks like a tree, cost d^{w+1} instead of d^n
- Modern solvers: learning, heuristics, restarts



a chain of small clusters:
treewidth 2, easy

(Cooper et al. 2020)

C. How solvers reason

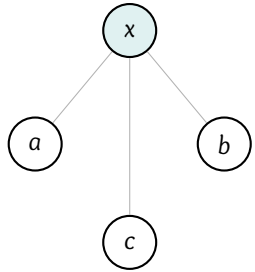


Can we learn and guarantee?

- Eliminate, propagate, branch
- Learn from your conflicts
- Branch and bound
- Live demo (Colab notebook)
- Compile once, query a thousand times
- What “NP-hard” means for you

Eliminating a variable replaces it by its consequences

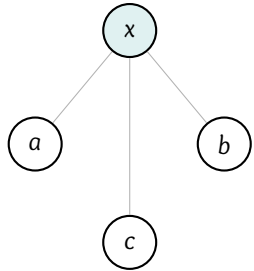
- Combine every function mentioning x, then make x disappear: $\exists$, $\min$ or $\sum$
- What remains is an *equivalent* problem without x: start again, until the answer
- No search, no backtracking — but the new function links together all the neighbours of x



- Cost: exponential in the *width* of the graph, not in the number of variables
- On clauses, combining is **resolution** (Davis–Putnam); and **propagation**, coming next, is its *bounded* version: anything above a fixed arity is thrown away

(Bertelè et al. 1972), (Dechter et al. 1994)

Eliminating a variable replaces it by its consequences

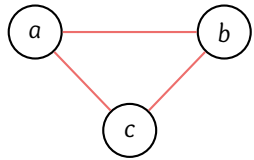


- Combine every function mentioning x, then make x disappear: $\exists$, $\min$ or $\sum$
- What remains is an *equivalent* problem without x: start again, until the answer
- No search, no backtracking — but the new function links together all the neighbours of x
- Cost: exponential in the *width* of the graph, not in the number of variables
- On clauses, combining is **resolution** (Davis–Putnam); and **propagation**, coming next, is its *bounded* version: anything above a fixed arity is thrown away

(Bertelè et al. 1972), (Dechter et al. 1994)

Eliminating a variable replaces it by its consequences

- Combine every function mentioning x, then make x disappear: $\exists$, $\min$ or $\sum$
- What remains is an *equivalent* problem without x: start again, until the answer
- No search, no backtracking — but the new function links together all the neighbours of x



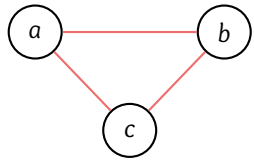
x eliminated:
its neighbours are linked

- Cost: exponential in the *width* of the graph, not in the number of variables
- On clauses, combining is **resolution** (Davis–Putnam); and **propagation**, coming next, is its *bounded* version: anything above a fixed arity is thrown away

(Bertelè et al. 1972), (Dechter et al. 1994)

Eliminating a variable replaces it by its consequences

- Combine every function mentioning x, then make x disappear: $\exists$, $\min$ or $\sum$
- What remains is an *equivalent* problem without x: start again, until the answer
- No search, no backtracking — but the new function links together all the neighbours of x



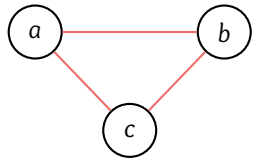
x eliminated:
its neighbours are linked

- Cost: exponential in the *width* of the graph, not in the number of variables
- On clauses, combining is **resolution** (Davis–Putnam); and **propagation**, coming next, is its *bounded* version: anything above a fixed arity is thrown away

(Bertelè et al. 1972), (Dechter et al. 1994)

Eliminating a variable replaces it by its consequences

- Combine every function mentioning x, then make x disappear: $\exists$, $\min$ or $\sum$
- What remains is an *equivalent* problem without x: start again, until the answer
- No search, no backtracking — but the new function links together all the neighbours of x



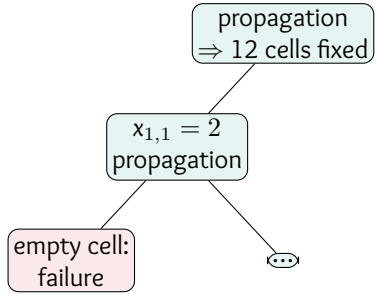
x eliminated:
its neighbours are linked

- Cost: exponential in the *width* of the graph, not in the number of variables
- On clauses, combining is **resolution** (Davis–Putnam); and **propagation**, coming next, is its *bounded* version: anything above a fixed arity is thrown away

(Bertelè et al. 1972), (Dechter et al. 1994)

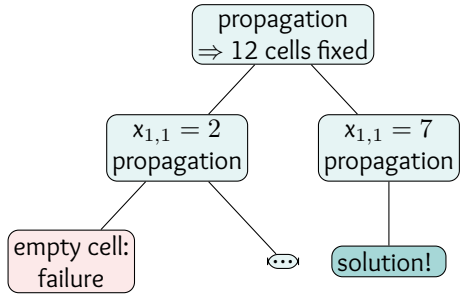
Propagate the consequences, then make a choice

- **Propagate:** remove impossible values (like a Sudoku player)
- **Branch:** when stuck, pick a cell and a value
- **Backtrack** when a cell has no value left



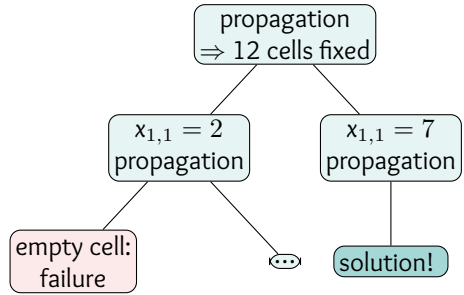
Propagate the consequences, then make a choice

- **Propagate:** remove impossible values (like a Sudoku player)
- **Branch:** when stuck, pick a cell and a value
- **Backtrack** when a cell has no value left



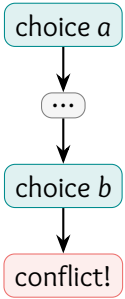
Propagate the consequences, then make a choice

- **Propagate:** remove impossible values (like a Sudoku player)
- **Branch:** when stuck, pick a cell and a value
- **Backtrack** when a cell has no value left



Learning from failure: the heart of SAT solvers

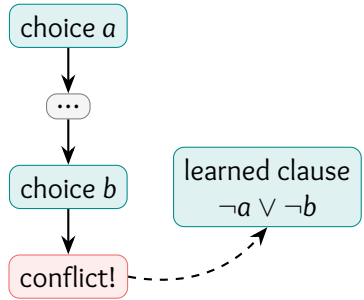
- Failure: analyse *which choices* caused it
- Derive a new clause from it: "never do $a \wedge b$ again"
- Jump straight back to the offending choice (non-chronological backtracking)
- CDCL: millions of clauses learned, forgotten, relearned



(Marques-Silva et al. 1999)

Learning from failure: the heart of SAT solvers

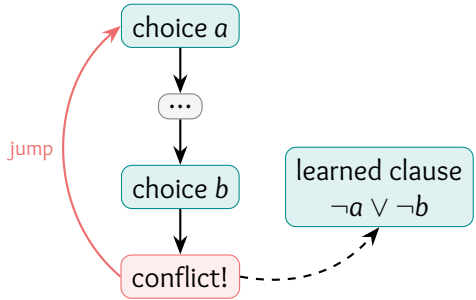
- Failure: analyse *which choices* caused it
- Derive a new clause from it: “never do $a \wedge b$ again”
- Jump straight back to the offending choice (non-chronological backtracking)
- CDCL: millions of clauses learned, forgotten, relearned



(Marques-Silva et al. 1999)

Learning from failure: the heart of SAT solvers

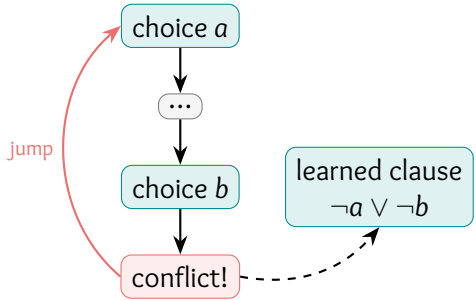
- Failure: analyse *which choices* caused it
- Derive a new clause from it: “never do $a \wedge b$ again”
- Jump straight back to the offending choice (non-chronological backtracking)
- CDCL: millions of clauses learned, forgotten, relearned



(Marques-Silva et al. 1999)

Learning from failure: the heart of SAT solvers

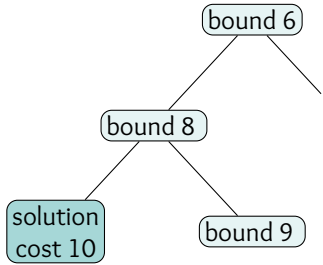
- Failure: analyse *which choices* caused it
- Derive a new clause from it: “never do $a \wedge b$ again”
- Jump straight back to the offending choice (non-chronological backtracking)
- CDCL: millions of clauses learned, forgotten, relearned



(Marques-Silva et al. 1999)

Branch and bound: proving the optimum

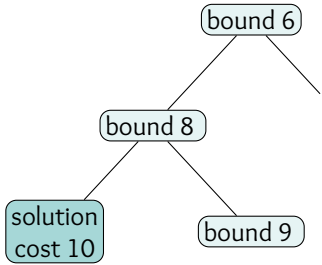
- Keep the best known solution (cost = 10)
- At each node: a *lower bound* on the achievable cost
- Bound $\geq$ 10: cut the branch (no point exploring it)
- CFN: we *move* costs between functions to push the bound up



(Cooper et al. 2020), (Allouche et al. 2015)

Branch and bound: proving the optimum

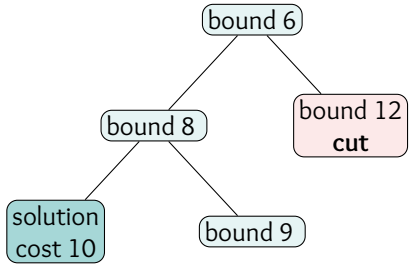
- Keep the best known solution (cost $= 10$)
- At each node: a *lower bound* on the achievable cost
- Bound ≥ 10: cut the branch (no point exploring it)
- CFN: we *move* costs between functions to push the bound up



(Cooper et al. 2020), (Allouche et al. 2015)

Branch and bound: proving the optimum

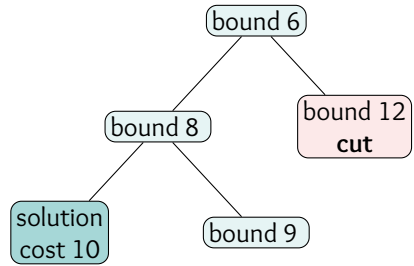
- Keep the best known solution (cost $=$ 10)
- At each node: a *lower bound* on the achievable cost
- Bound $\geq$ 10: cut the branch (no point exploring it)
- CFN: we *move* costs between functions to push the bound up



(Cooper et al. 2020), (Allouche et al. 2015)

Branch and bound: proving the optimum

- Keep the best known solution (cost $=$ 10)
- At each node: a *lower bound* on the achievable cost
- Bound $\geq$ 10: cut the branch (no point exploring it)
- CFN: we *move* costs between functions to push the bound up

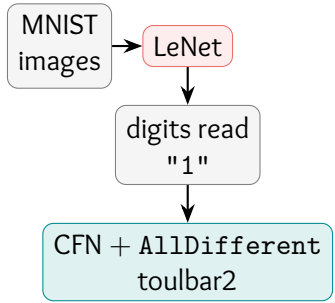


(Cooper et al. 2020), (Allouche et al. 2015)

Live demo: what if the Sudoku clues were images?

- A **LeNet** network trained on MNIST reads each handwritten digit
- The digits read become the clues, `AllDifferent` on rows, columns and boxes
- Grid 2: correct reading → solved and *proved*
- Grid 0: a 2 read as a 1 → **no solution**

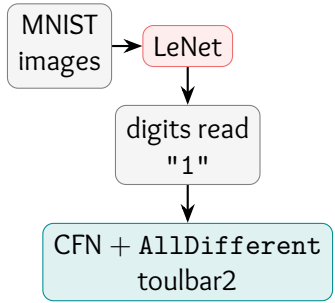
Colab notebook, nothing to install:
`colab.research.google.com/drive/1ew7Icel…`



Live demo: what if the Sudoku clues were images?

- A **LeNet** network trained on MNIST reads each handwritten digit
- The digits read become the clues, `AllDifferent` on rows, columns and boxes
- Grid 2: correct reading → solved and *proved*
- Grid 0: a 2 read as a 1 → **no solution**

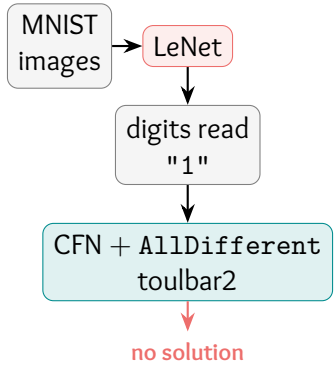
Colab notebook, nothing to install:
`colab.research.google.com/drive/1ew7Icel…`



Live demo: what if the Sudoku clues were images?

- A **LeNet** network trained on MNIST reads each handwritten digit
- The digits read become the clues, `AllDifferent` on rows, columns and boxes
- Grid 2: correct reading → solved and *proved*
- Grid 0: a **2** read as a **1** → **no solution**

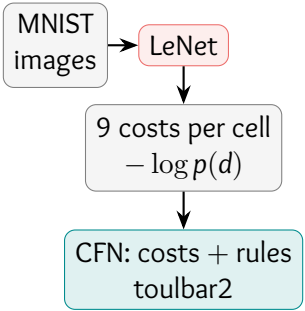
Colab notebook, nothing to install:
`colab.research.google.com/drive/1ew7Icel…`



The network hesitates, the solver decides — and fixes its reading

- A clue is no longer imposed: LeNet hands over its **9 costs** $-\log p(d)$
- The rules stay hard: cost $K = 10^6$
- Minimising the total = looking for the most *probable* grid that obeys the rules
- Grid 0: solved, and the misread 1 is replaced by the right 2

One function of the notebook changed (`set_hint`); the network was not touched.

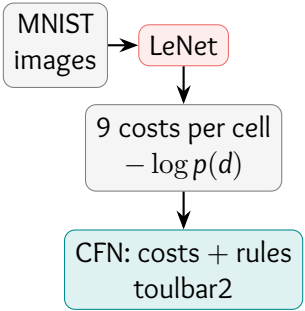


corrected solution

The network hesitates, the solver decides — and fixes its reading

- A clue is no longer imposed: LeNet hands over its **9 costs** $-\log p(d)$
- The rules stay hard: cost $K = 10^6$
- Minimising the total = looking for the most *probable* grid that obeys the rules
- Grid 0: solved, and the misread 1 is replaced by the right 2

One function of the notebook changed (`set_hint`); the network was not touched.

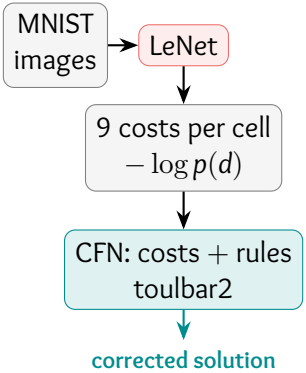


corrected solution

The network hesitates, the solver decides — and fixes its reading

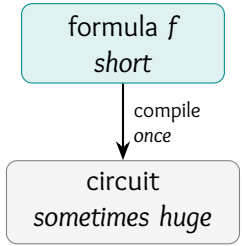
- A clue is no longer imposed: LeNet hands over its **9 costs** $-\log p(d)$
- The rules stay hard: cost $K = 10^6$
- Minimising the total = looking for the most *probable* grid that obeys the rules
- Grid 0: solved, and the misread **1** is replaced by the right **2**

One function of the notebook changed (`set_hint`); the network was not touched.



Compile once, query a thousand times

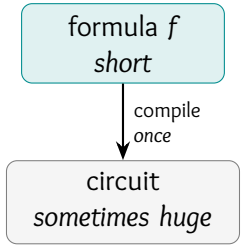
- So far, the solver starts from scratch at every question
- Another strategy: turn f *once* into a **circuit** — binary decision diagram (BDD), d-DNNF, SDD
- Each query is then just a traversal: *linear* in the size of the circuit
- The price: the circuit may blow up — and the more compact a language, the fewer questions it answers in polynomial time



(Darwiche et al. 2002)

Compile once, query a thousand times

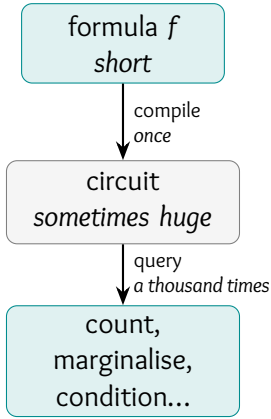
- So far, the solver starts from scratch at every question
- Another strategy: turn *f once* into a **circuit** — binary decision diagram (BDD), d-DNNF, SDD
- Each query is then just a traversal: *linear* in the size of the circuit
- The price: the circuit may blow up — and the more compact a language, the fewer questions it answers in polynomial time



(Darwiche et al. 2002)

Compile once, query a thousand times

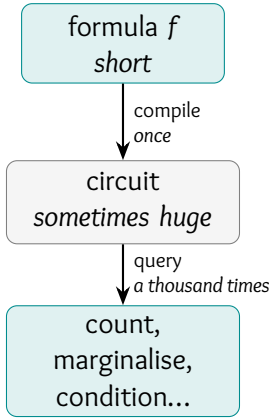
- So far, the solver starts from scratch at every question
- Another strategy: turn *f once* into a **circuit** — binary decision diagram (BDD), d-DNNF, SDD
- Each query is then just a traversal: *linear* in the size of the circuit
- The price: the circuit may blow up — and the more compact a language, the fewer questions it answers in polynomial time



(Darwiche et al. 2002)

Compile once, query a thousand times

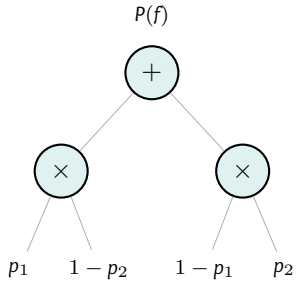
- So far, the solver starts from scratch at every question
- Another strategy: turn *f once* into a **circuit** — binary decision diagram (BDD), d-DNNF, SDD
- Each query is then just a traversal: *linear* in the size of the circuit
- The price: the circuit may blow up — and the more compact a language, the fewer questions it answers in polynomial time



(Darwiche et al. 2002)

The same circuit, made probabilistic — and differentiable

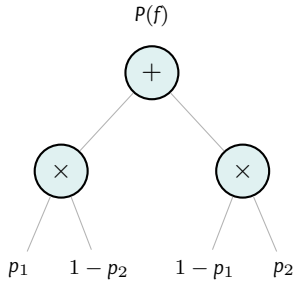
- Replace $\vee$ by $+$ and $\wedge$ by $\times$: the circuit no longer returns true or false, but a probability
- Same structural properties $\Rightarrow$ *exact* marginals and normalisation, in linear time
- An arithmetic circuit can be **differentiated**: evaluating and backpropagating cost the same
- A compiled logical constraint thus becomes a *differentiable loss term* (part F)



(Darwiche 2003)

The same circuit, made probabilistic — and differentiable

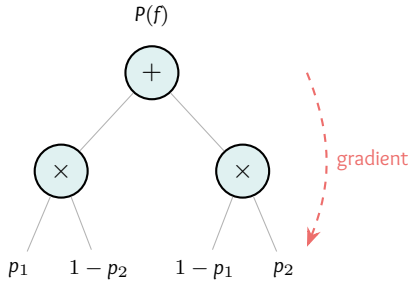
- Replace $\vee$ by $+$ and $\wedge$ by $\times$: the circuit no longer returns true or false, but a probability
- Same structural properties $\Rightarrow$ *exact* marginals and normalisation, in linear time
- An arithmetic circuit can be **differentiated**: evaluating and backpropagating cost the same
- A compiled logical constraint thus becomes a *differentiable loss term* (part F)



(Darwiche 2003)

The same circuit, made probabilistic — and differentiable

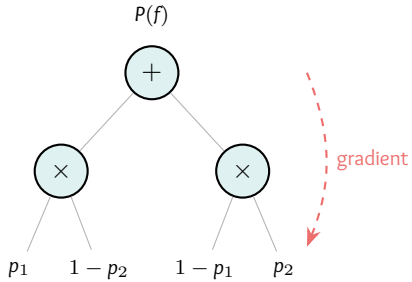
- Replace $\vee$ by $+$ and $\wedge$ by $\times$: the circuit no longer returns true or false, but a probability
- Same structural properties $\Rightarrow$ *exact* marginals and normalisation, in linear time
- An arithmetic circuit can be **differentiated**: evaluating and backpropagating cost the same
- A compiled logical constraint thus becomes a *differentiable loss term* (part F)



(Darwiche 2003)

The same circuit, made probabilistic — and differentiable

- Replace $\vee$ by $+$ and $\wedge$ by $\times$: the circuit no longer returns true or false, but a probability
- Same structural properties $\Rightarrow$ *exact* marginals and normalisation, in linear time
- An arithmetic circuit can be **differentiated**: evaluating and backpropagating cost the same
- A compiled logical constraint thus becomes a *differentiable loss term* (part F)



(Darwiche 2003)

NP-hard: what it means for you

1. **Do not give up**: try a generic solver on *your* instances
2. **Do not reinvent**: a naive in-house code will almost always be beaten
3. **Modelling matters**: a good model (and the right formalism) changes everything
4. **Accept trade-offs**: a time limit, a solution + a bound (guaranteed gap), or a heuristic
5. **Know what you have**: a *proved* optimal solution, or merely a "good" one?

NP-hard: what it means for you

1. **Do not give up**: try a generic solver on *your* instances
2. **Do not reinvent**: a naive in-house code will almost always be beaten
3. **Modelling matters**: a good model (and the right formalism) changes everything
4. **Accept trade-offs**: a time limit, a solution + a bound (guaranteed gap), or a heuristic
5. **Know what you have**: a *proved* optimal solution, or merely a "good" one?

NP-hard: what it means for you

1. **Do not give up**: try a generic solver on *your* instances
2. **Do not reinvent**: a naive in-house code will almost always be beaten
3. **Modelling matters**: a good model (and the right formalism) changes everything
4. **Accept trade-offs**: a time limit, a solution + a bound (guaranteed gap), or a heuristic
5. **Know what you have**: a *proved* optimal solution, or merely a "good" one?

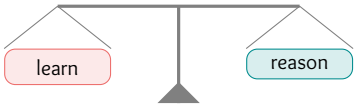
NP-hard: what it means for you

1. **Do not give up**: try a generic solver on *your* instances
2. **Do not reinvent**: a naive in-house code will almost always be beaten
3. **Modelling matters**: a good model (and the right formalism) changes everything
4. **Accept trade-offs**: a time limit, a solution + a bound (guaranteed gap), or a heuristic
5. **Know what you have**: a *proved* optimal solution, or merely a "good" one?

NP-hard: what it means for you

1. **Do not give up**: try a generic solver on *your* instances
2. **Do not reinvent**: a naive in-house code will almost always be beaten
3. **Modelling matters**: a good model (and the right formalism) changes everything
4. **Accept trade-offs**: a time limit, a solution + a bound (guaranteed gap), or a heuristic
5. **Know what you have**: a *proved* optimal solution, or merely a "good" one?

D. Neuro-symbolic AI: what are we talking about?



Can we learn and guarantee?

- Why mix them
- Kautz's taxonomy (6 types)
- More recent classifications
- Where my own work sits

Neurons and symbols have complementary strengths

	Neural	Symbolic
Natural inputs (images, sequences)	+	–
Learning from data	+	–
Guarantees, proofs	–	+
Explicit knowledge, control	–	+
Long, combinatorial reasoning	–	+

- Kautz: a fast, intuitive “System 1” / a slow, deliberate “System 2” (Kahneman)
- Garcez & Lamb: unite the robust learning of networks with the explainable reasoning of symbols

(Kautz 2022), (Garcez et al. 2023)

Neurons and symbols have complementary strengths

	Neural	Symbolic
Natural inputs (images, sequences)	+	–
Learning from data	+	–
Guarantees, proofs	–	+
Explicit knowledge, control	–	+
Long, combinatorial reasoning	–	+

- Kautz: a fast, intuitive “System 1” / a slow, deliberate “System 2” (Kahneman)
- Garcez & Lamb: unite the robust learning of networks with the explainable reasoning of symbols

(Kautz 2022), (Garcez et al. 2023)

Neurons and symbols have complementary strengths

	Neural	Symbolic
Natural inputs (images, sequences)	+	–
Learning from data	+	–
Guarantees, proofs	–	+
Explicit knowledge, control	–	+
Long, combinatorial reasoning	–	+

- Kautz: a fast, intuitive “System 1” / a slow, deliberate “System 2” (Kahneman)
- Garcez & Lamb: unite the robust learning of networks with the explainable reasoning of symbols

(Kautz 2022), (Garcez et al. 2023)

Kautz: six ways of assembling neurons and symbols (1/2)

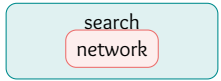
1. Symbolic Neuro symbolic



Symbols in and out, a network in between

E.g. standard natural language processing

2. Symbolic[Neuro]



A symbolic solver calls a network as a subroutine

E.g. AlphaGo (tree search + evaluation network)

3. Neuro | Symbolic



The network produces a symbolic structure, handled by a reasoner

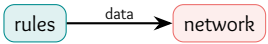
E.g. the Neuro-Symbolic Concept Learner

Tools: `DeepProbLog`, `Scallop`, `NeurASP`

(Kautz 2022)

Kautz: six ways of assembling neurons and symbols (2/2)

4. Neuro:Symbolic→Neuro



Symbolic rules generate the training examples

E.g. simplifying mathematical expressions (Lample & Charton)

5. Neuro$_{\text{Symbolic}}$

rules ⇒ network

constrained network

The rules are compiled into the network: architecture or loss function

E.g. Logic Tensor Networks, PINNs

Tools: `LTN`, `PySDD` (semantic loss)

6. Neuro[Symbolic]

network

reasoner

The network decides when to call a reasoner (System 2)

E.g. an agent that fires a theorem prover; the most promising route according to Kautz

(Kautz 2022)

More recent, finer-grained classifications

Marra et al. (Artificial Intelligence, 2024)

From statistical relational to neuro-symbolic: 7 dimensions

1. type of logical inference
2. syntax of the logical theories
3. logical semantics
4. what is learned (parameters, structure)
5. symbolic / sub-symbolic representations
6. degree of fidelity to the original paradigms
7. target learning tasks

Bhuyan et al. (2024)

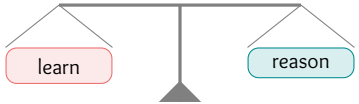
A recent survey: representation, learning, reasoning, decision

Garcez & Lamb (2023)

The "3rd wave": trust, explainability, knowledge extraction

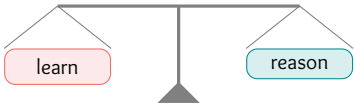
(Marra et al. 2024), (Bhuyan et al. 2024), (Garcez et al. 2023)

Break — 10 minutes



So far: *reasoning*, and the ways of plugging learning into it.
After the break: *learned* costs, on a real protein problem.

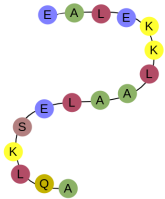
E. Case study: designing proteins



Can we learn and guarantee?

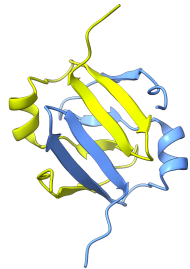
- Protein design is a CFN
- The rules of physics, and the search without guarantees
- What annealing misses, and what can be proved
- The turning point: learn the costs instead of writing them

A protein is a chain that folds in order to work



amino acid sequence





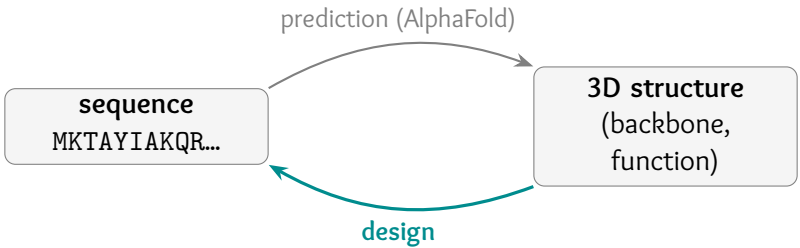
3D structure

- 20 natural amino acids
- The sequence determines the shape
- The shape determines the function (catalysis, binding…)

Applications: health (antibodies, vaccines), green chemistry (enzymes), biomaterials

Figure: earlier slide deck; structure: DPBB barrel

Design is an inverse problem, like a Sudoku



Sudoku

- 81 cells, 9 digits
- fixed grid geometry
- known rules

Protein

- *n* positions (up to hundreds), 20 amino acids
- given backbone geometry
- *approximate* rules: physics

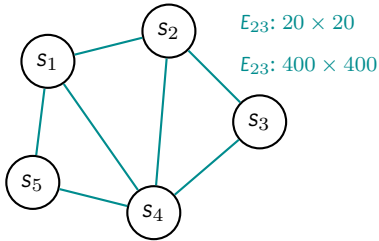
(Defresne et al. 2026a)

Designing a sequence = minimising a CFN

- Variable s_i: the amino acid at position i
- Pairwise score (Potts model):

$$E(s \mid B) = \sum_{i<j} E_{ij}(s_i, s_j \mid B)$$

- Best sequence = minimal cost
  = MAP of $P(s \mid B) \propto e^{-E(s|B)}$
- NP-hard, yet toulbar2 optimises it
  exactly on real instances



an edge = two positions close in space

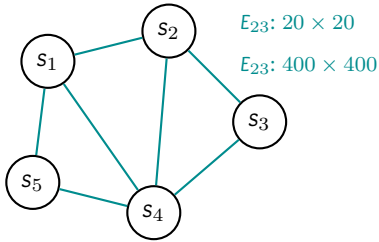
(Pierce et al. 2002), (Allouche et al. 2014)

Designing a sequence = minimising a CFN

- Variable s_i: the amino acid at position i
- Pairwise score (Potts model):

$$E(s \mid B) = \sum_{i<j} E_{ij}(s_i, s_j \mid B)$$

- Best sequence = minimal cost
  = MAP of $P(s \mid B) \propto e^{-E(s|B)}$
- NP-hard, yet toulbar2 optimises it *exactly* on real instances



an edge = two positions close in space

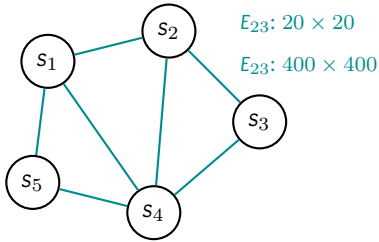
(Pierce et al. 2002), (Allouche et al. 2014)

Designing a sequence = minimising a CFN

- Variable s_i: the amino acid at position i
- Pairwise score (Potts model):

$$E(s \mid B) = \sum_{i<j} E_{ij}(s_i, s_j \mid B)$$

- Best sequence = minimal cost
  = MAP of $P(s \mid B) \propto e^{-E(s|B)}$
- NP-hard, yet toulbar2 optimises it *exactly* on real instances

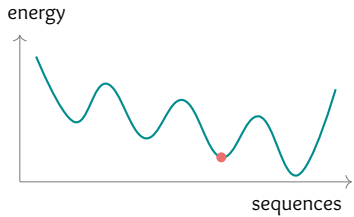


an edge = two positions close in space

(Pierce et al. 2002), (Allouche et al. 2014)

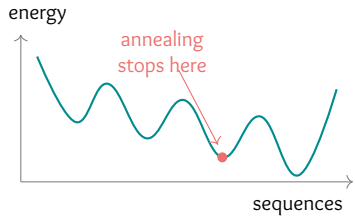
The usual way: physics writes the costs, annealing searches

- The costs E_{ij} come from physical terms: van der Waals, electrostatics, solvation, hydrogen bonds
- Conformations discretised into *rotamers*: this really is a CFN
- Search: **simulated annealing**, restarted a great many times (Rosetta)
- **Symbolic, yes. Automated reasoning, no**: no bound, no proof, never "there is nothing better"



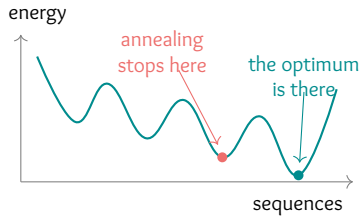
The usual way: physics writes the costs, annealing searches

- The costs E_{ij} come from physical terms: van der Waals, electrostatics, solvation, hydrogen bonds
- Conformations discretised into *rotamers*: this really is a CFN
- Search: **simulated annealing**, restarted a great many times (Rosetta)
- **Symbolic, yes. Automated reasoning, no:** no bound, no proof, never "there is nothing better"



The usual way: physics writes the costs, annealing searches

- The costs E_{ij} come from physical terms: van der Waals, electrostatics, solvation, hydrogen bonds
- Conformations discretised into *rotamers*: this really is a CFN
- Search: **simulated annealing**, restarted a great many times (Rosetta)
- **Symbolic, yes. Automated reasoning, no**: no bound, no proof, never "there is nothing better"



toulbar2 proves the optimum — and measures what annealing misses

- Same energy (Talaris2014), same rotamers, an *exact* method: branch and bound, consistencies, tree decomposition
- Global optimum **proved** on full redesigns, in spaces of up to 10^{234} — a single core, at most 66 GB

- Against it, annealing: the probability of reaching the optimum **drops fast towards 0**
- Worst case: despite **1,000 restarts**, still more than one Rosetta unit away from the optimum
- $\Rightarrow$ sequences differing from the optimal one in more than **30 % of their amino acids**

(Simoncini et al. 2015)

toulbar2 proves the optimum — and measures what annealing misses

- Same energy (Talaris2014), same rotamers, an *exact* method: branch and bound, consistencies, tree decomposition
- Global optimum **proved** on full redesigns, in spaces of up to 10^{234} — a single core, at most 66 GB

- Against it, annealing: the probability of reaching the optimum **drops fast towards 0**
- Worst case: despite **1,000 restarts**, still more than one Rosetta unit away from the optimum
- $\Rightarrow$ sequences differing from the optimal one in more than **30 % of their amino acids**

(Simoncini et al. 2015)

toulbar2 proves the optimum — and measures what annealing misses

- Same energy (Talaris2014), same rotamers, an *exact* method: branch and bound, consistencies, tree decomposition
- Global optimum **proved** on full redesigns, in spaces of up to 10^{234} — a single core, at most 66 GB

- Against it, annealing: the probability of reaching the optimum **drops fast towards 0**
- Worst case: despite **1,000 restarts**, still more than one Rosetta unit away from the optimum
- $\Rightarrow$ sequences differing from the optimal one in more than **30 % of their amino acids**

(Simoncini et al. 2015)

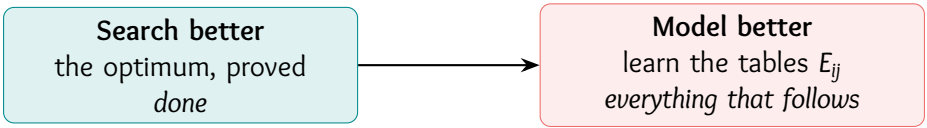
toulbar2 proves the optimum — and measures what annealing misses

- Same energy (Talaris2014), same rotamers, an *exact* method: branch and bound, consistencies, tree decomposition
- Global optimum **proved** on full redesigns, in spaces of up to 10^{234} — a single core, at most 66 GB

- Against it, annealing: the probability of reaching the optimum **drops fast towards 0**
- Worst case: despite **1,000 restarts**, still more than one Rosetta unit away from the optimum
- $\Rightarrow$ sequences differing from the optimal one in more than **30 % of their amino acids**

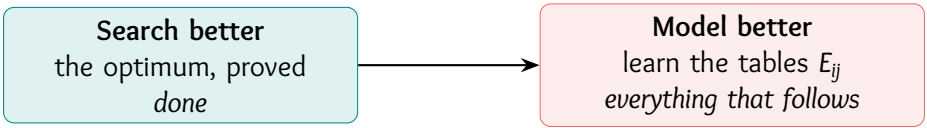
(Simoncini et al. 2015)

Doing better: searching better is not enough, the model must change



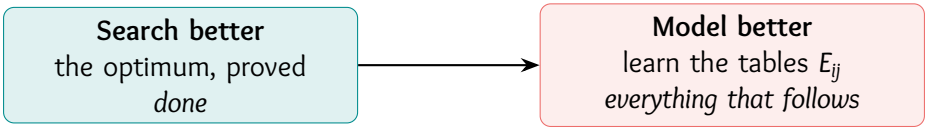
- An exact solver optimises an approximate energy *exactly*: it cannot fix the model
- And that energy rests on nineteenth- and early twentieth-century physics
- Experimental structures, by contrast, are plentiful and public (PDB)

Doing better: searching better is not enough, the model must change



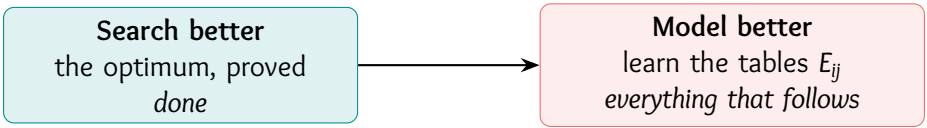
- An exact solver optimises an approximate energy *exactly*: it cannot fix the model
- And that energy rests on nineteenth- and early twentieth-century physics
- Experimental structures, by contrast, are plentiful and public (PDB)

Doing better: searching better is not enough, the model must change



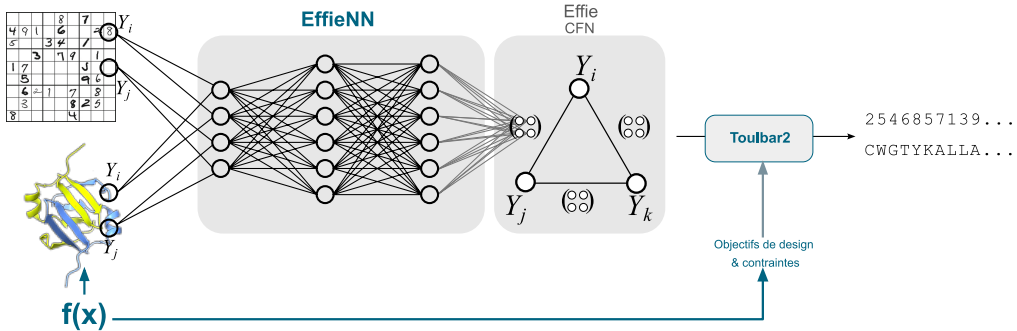
- An exact solver optimises an approximate energy *exactly*: it cannot fix the model
- And that energy rests on nineteenth- and early twentieth-century physics
- Experimental structures, by contrast, are plentiful and public (PDB)

Doing better: searching better is not enough, the model must change



- An exact solver optimises an approximate energy *exactly*: it cannot fix the model
- And that energy rests on nineteenth- and early twentieth-century physics
- Experimental structures, by contrast, are plentiful and public (PDB)

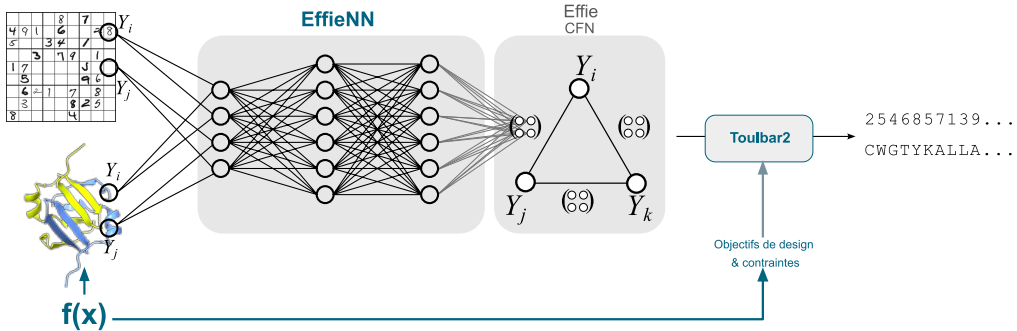
EffieDes: a network writes the CFN, toulbar2 solves it



- EffieNN: reads the backbone B, predicts the tables E_{ij} (20×20) of pairs within 15 Å
- Trained on structures from the PDB (Protein Data Bank)
- Optimisation: exact (toulbar2) or relaxed (LR-BCD) for large proteins

Figure: (Defresne et al. 2026a), Fig. 2 (adapted)

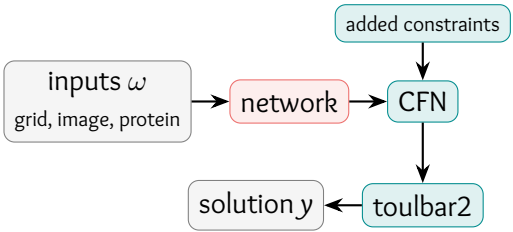
EffieDes: a network writes the CFN, toulbar2 solves it



- EffieNN: reads the backbone B, predicts the tables E_{ij} (20×20) of pairs within 15 Å
- Trained on structures from the PDB (Protein Data Bank)
- Optimisation: exact (toulbar2) or relaxed (LR-BCD) for large proteins

Figure: (Defresne et al. 2026a), Fig. 2 (adapted)

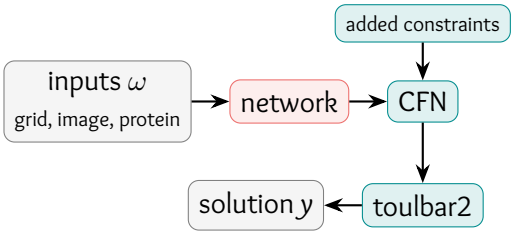
Where do Effie and EffieDes sit?



- Type 3, **Neuro | Symbolic**: the network *writes* a symbolic model (a CFN)
- With symbolic knowledge injected at inference time
- Limits of the taxonomy: it describes the architecture, not the learning signal nor the guarantees; the types overlap

(Defresne et al. 2026a), (Defresne et al. 2026b)

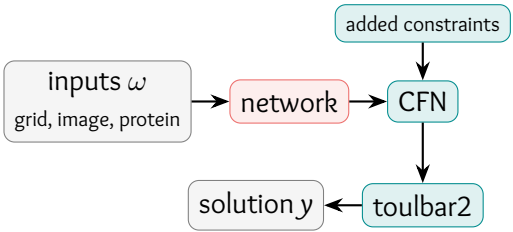
Where do Effie and EffieDes sit?



- Type 3, **Neuro | Symbolic**: the network *writes* a symbolic model (a CFN)
- With symbolic knowledge injected at inference time
- Limits of the taxonomy: it describes the architecture, not the learning signal nor the guarantees; the types overlap

(Defresne et al. 2026a), (Defresne et al. 2026b)

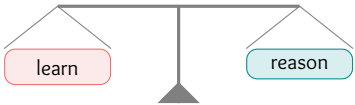
Where do Effie and EffieDes sit?



- Type 3, **Neuro | Symbolic**: the network *writes* a symbolic model (a CFN)
- With symbolic knowledge injected at inference time
- Limits of the taxonomy: it describes the architecture, not the learning signal nor the guarantees; the types overlap

(Defresne et al. 2026a), (Defresne et al. 2026b)

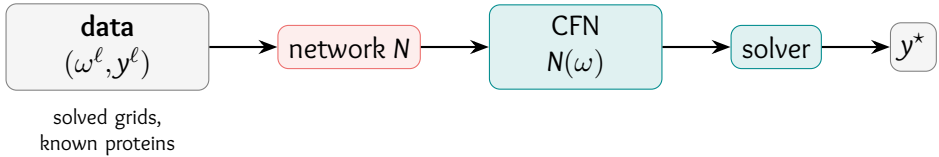
F. Learning the rules themselves



Can we learn and guarantee?

- Learning a CFN without a solver
- The redundancy problem
- The Emmental, and what it gives
- What it gives on proteins

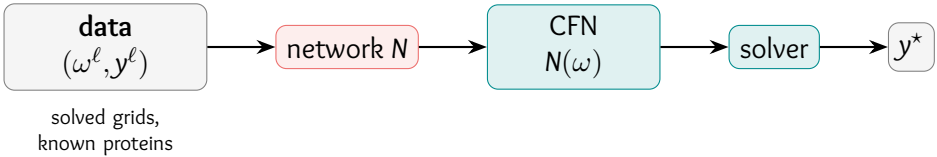
The problem: learning to solve without knowing the rules



- We observe *solutions*, not the rules that govern them
- Sudoku: learning to play without being told the rules
- Proteins: nobody knows the exact "rules" of folding

(Defresne et al. 2026a)

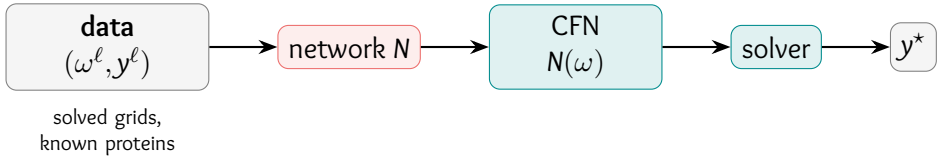
The problem: learning to solve without knowing the rules



- We observe *solutions*, not the rules that govern them
- Sudoku: learning to play without being told the rules
- Proteins: nobody knows the exact "rules" of folding

(Defresne et al. 2026a)

The problem: learning to solve without knowing the rules

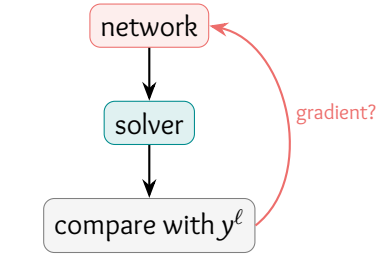


- We observe *solutions*, not the rules that govern them
- Sudoku: learning to play without being told the rules
- Proteins: nobody knows the exact "rules" of folding

(Defresne et al. 2026a)

Putting the solver inside the learning loop is very expensive

- The natural idea: solve, compare with y^ℓ, backpropagate
- **Obstacle 1:** the output is discrete, the gradient is zero or undefined
- **Obstacle 2:** one NP-hard call *per example and per epoch*
- **Obstacle 3:** in the early epochs the predicted models are random, hence among the hardest to solve

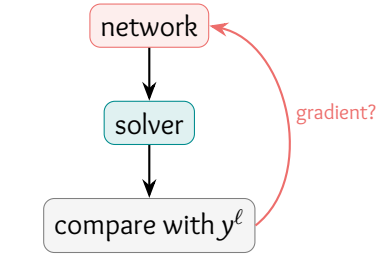


$\times$ 100,000 examples $\times$ 50 epochs

(Defresne et al. 2026a), (Defresne et al. 2023)

Putting the solver inside the learning loop is very expensive

- The natural idea: solve, compare with y^ℓ, backpropagate
- **Obstacle 1**: the output is discrete, the gradient is zero or undefined
- **Obstacle 2**: one NP-hard call *per example and per epoch*
- **Obstacle 3**: in the early epochs the predicted models are random, hence among the hardest to solve

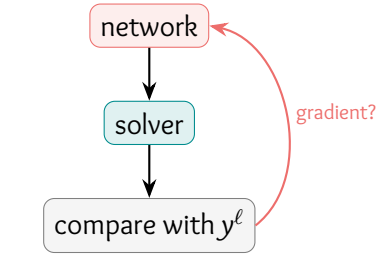


$\times$ 100,000 examples $\times$ 50 epochs

(Defresne et al. 2026a), (Defresne et al. 2023)

Putting the solver inside the learning loop is very expensive

- The natural idea: solve, compare with y^ℓ, backpropagate
- **Obstacle 1**: the output is discrete, the gradient is zero or undefined
- **Obstacle 2**: one NP-hard call *per example and per epoch*
- **Obstacle 3**: in the early epochs the predicted models are random, hence among the hardest to solve

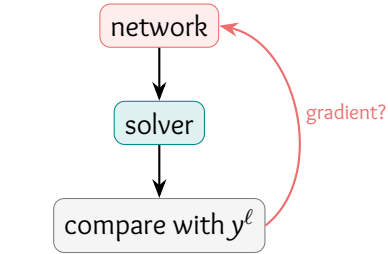


$\times$ 100,000 examples $\times$ 50 epochs

(Defresne et al. 2026a), (Defresne et al. 2023)

Putting the solver inside the learning loop is very expensive

- The natural idea: solve, compare with y^ℓ, backpropagate
- **Obstacle 1**: the output is discrete, the gradient is zero or undefined
- **Obstacle 2**: one NP-hard call *per example and per epoch*
- **Obstacle 3**: in the early epochs the predicted models are random, hence among the hardest to solve



(Defresne et al. 2026a), (Defresne et al. 2023)

Pseudo-likelihood takes the solver out of the loop

- A CFN defines a probability: $P(y) \propto \exp(-\text{cost}(y))$
- Exact likelihood: needs the normalising constant Z #P-hard
- **Pseudo-likelihood** (Besag, 1975): predict *one* variable only, the others being observed

$$\text{NPLL} = -\sum_{\ell}\sum_{i} \log P\big(y_i^{\ell} \mid y_{-i}^{\ell}\big) \qquad P(Y_i \mid y_{-i}) = \text{softmax}\Big(-\sum_{j\neq i} N(\omega)[i,j](Y_i, y_j)\Big)$$

- A *local* computation: a sum over the neighbours, then a softmax — no solver call
- Cost $O(n^2 d)$ per example, vectorisable; statistically consistent

(Besag 1975), (Defresne et al. 2026a)

Pseudo-likelihood takes the solver out of the loop

- A CFN defines a probability: $P(y) \propto \exp(-\text{cost}(y))$
- Exact likelihood: needs the normalising constant Z #P-hard
- **Pseudo-likelihood** (Besag, 1975): predict *one* variable only, the others being observed

$$\text{NPLL} = -\sum_{\ell}\sum_{i} \log P\big(y_i^{\ell} \mid y_{-i}^{\ell}\big) \qquad P(Y_i \mid y_{-i}) = \text{softmax}\Big(-\sum_{j\neq i} N(\omega)[i,j](Y_i, y_j)\Big)$$

- A *local* computation: a sum over the neighbours, then a softmax — no solver call
- Cost $O(n^2 d)$ per example, vectorisable; statistically consistent

(Besag 1975), (Defresne et al. 2026a)

Pseudo-likelihood takes the solver out of the loop

- A CFN defines a probability: $P(y) \propto \exp(-\text{cost}(y))$
- Exact likelihood: needs the normalising constant Z #P-hard
- **Pseudo-likelihood** (Besag, 1975): predict *one* variable only, the others being observed

$$\text{NPLL} = -\sum_{\ell} \sum_{i} \log P\big(y_i^{\ell} \mid y_{-i}^{\ell}\big) \qquad P(Y_i \mid y_{-i}) = \text{softmax}\Big(-\sum_{j \neq i} N(\omega)[i,j](Y_i, y_j)\Big)$$

- A *local* computation: a sum over the neighbours, then a softmax — no solver call
- Cost $O(n^2 d)$ per example, vectorisable; statistically consistent

(Besag 1975), (Defresne et al. 2026a)

Pseudo-likelihood takes the solver out of the loop

- A CFN defines a probability: $P(y) \propto \exp(-\text{cost}(y))$
- Exact likelihood: needs the normalising constant Z #P-hard
- **Pseudo-likelihood** (Besag, 1975): predict *one* variable only, the others being observed

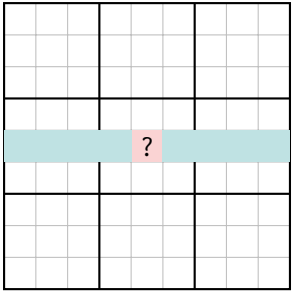
$$\text{NPLL} = -\sum_{\ell} \sum_{i} \log P\big(y_i^{\ell} \mid y_{-i}^{\ell}\big) \qquad P(Y_i \mid y_{-i}) = \text{softmax}\Big(-\sum_{j \neq i} N(\omega)[i,j](Y_i, y_j) \Big)$$

- A *local* computation: a sum over the neighbours, then a softmax — no solver call
- Cost $O(n^2 d)$ per example, vectorisable; statistically consistent

(Besag 1975), (Defresne et al. 2026a)

But pseudo-likelihood does not learn all the rules

- Sudoku: to recover one hidden cell, the *row* constraints are enough
- The gradient of the column and box constraints vanishes: they are *redundant in that context*
- But at inference time, all the rules are needed!
- Result: 0 % of grids solved

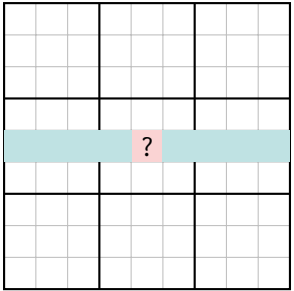


the row alone is enough to guess the masked cell

(Montanari et al. 2009), (Defresne et al. 2026a)

But pseudo-likelihood does not learn all the rules

- Sudoku: to recover one hidden cell, the *row* constraints are enough
- The gradient of the column and box constraints vanishes: they are *redundant in that context*
- But at inference time, all the rules are needed!
- Result: 0 % of grids solved

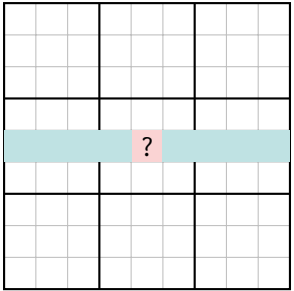


the row alone is enough to guess the masked cell

(Montanari et al. 2009), (Defresne et al. 2026a)

But pseudo-likelihood does not learn all the rules

- Sudoku: to recover one hidden cell, the *row* constraints are enough
- The gradient of the column and box constraints vanishes: they are *redundant in that context*
- But at inference time, all the rules are needed!
- Result: 0 % of grids solved

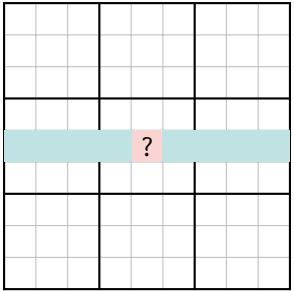


the row alone is enough to guess the masked cell

(Montanari et al. 2009), (Defresne et al. 2026a)

But pseudo-likelihood does not learn all the rules

- Sudoku: to recover one hidden cell, the *row* constraints are enough
- The gradient of the column and box constraints vanishes: they are *redundant in that context*
- But at inference time, all the rules are needed!
- Result: 0 % of grids solved

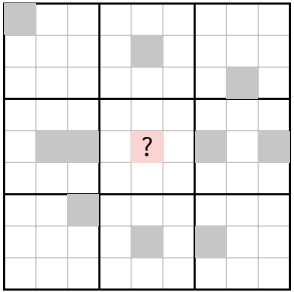


the row alone is enough to guess the masked cell

(Montanari et al. 2009), (Defresne et al. 2026a)

The Emmental: punching holes to force learning

- The idea: *also* mask k random neighbours, every time
- Without the row, the column and the box are needed to guess: the gradient is back
- **E-PLL** (*Emmental* pseudo-log-likelihood): k random holes
- Inspired by *dropout*; robust to k (except $k = 0$ or $k \simeq n$)

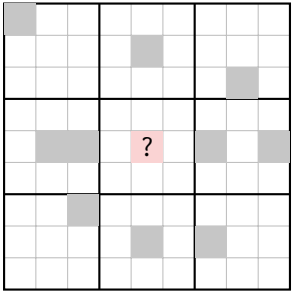


k extra cells ignored (in grey) at each step

(Defresne et al. 2026a)

The Emmental: punching holes to force learning

- The idea: *also* mask k random neighbours, every time
- Without the row, the column and the box are needed to guess: the gradient is back
- **E-PLL** (*Emmental* pseudo-log-likelihood): k random holes
- Inspired by *dropout*; robust to k (except $k = 0$ or $k \simeq n$)

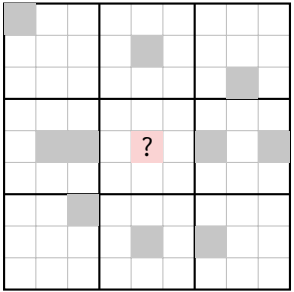


k extra cells ignored (in grey) at each step

(Defresne et al. 2026a)

The Emmental: punching holes to force learning

- The idea: *also* mask k random neighbours, every time
- Without the row, the column and the box are needed to guess: the gradient is back
- **E-PLL** (*Emmental* pseudo-log-likelihood): k random holes
- Inspired by *dropout*; robust to k (except $k = 0$ or $k \simeq n$)

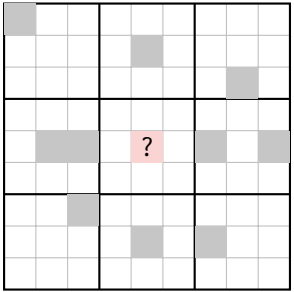


k extra cells ignored (in grey) at each step

(Defresne et al. 2026a)

The Emmental: punching holes to force learning

- The idea: *also* mask k random neighbours, every time
- Without the row, the column and the box are needed to guess: the gradient is back
- **E-PLL** (*Emmental* pseudo-log-likelihood): k random holes
- Inspired by *dropout*; robust to k (except $k = 0$ or $k \simeq n$)



k extra cells ignored (in grey) at each step

(Defresne et al. 2026a)

100 grids are enough for a perfect Sudoku

Type	Approach	Acc.	#clues	Training	Time (h)
DL	RRN	96.6 %	17	180,000	> 50
	Rec. Trans.	96.7 %	17	180,000	> 50
	DDPM	99.2–100 %	33.8	100,000	13.6
	DDPM	0.2 %	17	–	–
	HRM	55 %	24.8	$1{,}000 \times 1{,}001$	> 10
Relax. + DL	SATNet	95.1–99.8 %	36.2	9,000	2.9
	SATNet	86.1–86.2 %	17	–	–
Reason. + DL	Hinge (embedded solver)	**100** %	17	1,000	> 50
	E-PLL	**100** %	17	**100**	**0.05**
	E-PLL	**100** %	24.8	$\mathbf{10 \times 100}$	**0.04**

Fewer clues = harder. 17 is the minimum for a unique solution.

Figure: (Defresne et al. 2026a), Tab. 2

The learned rules can be read

- The learned model is a cost table, not a black box
- After hardening: exactly the **810** binary constraints of Sudoku, no more, no less
- So the 100 % observed on the test set extends to *every* grid

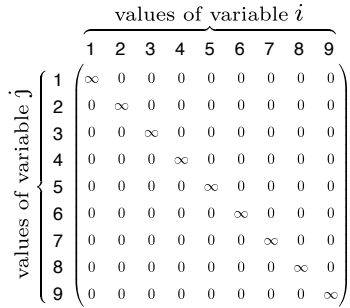
$$
\begin{array}{c}
 \\
\text{values of variable } j
\end{array}
\begin{array}{c}
\overbrace{}^{\text{values of variable } i} \\
1\;2\;3\;4\;5\;6\;7\;8\;9
\end{array}
\begin{array}{c}
\\
1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9
\end{array}
\begin{pmatrix}
\infty & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & \infty & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & \infty & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & \infty & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & \infty & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & \infty & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & \infty & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & \infty & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \infty
\end{pmatrix}
$$

one learned rule: ∞ costs on the diagonal

Figure: (Defresne et al. 2026a), Fig. 3

The learned rules can be read

- The learned model is a cost table, not a black box
- After hardening: exactly the **810** binary constraints of Sudoku, no more, no less
- So the 100 % observed on the test set extends to *every* grid



one learned rule: ∞ costs on the diagonal

Figure: (Defresne et al. 2026a), Fig. 3

The learned rules can be read

- The learned model is a cost table, not a black box
- After hardening: exactly the **810** binary constraints of Sudoku, no more, no less
- So the 100 % observed on the test set extends to *every* grid

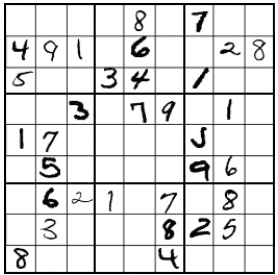
values of variable i

values of variable j	1	2	3	4	5	6	7	8	9
1	∞	0	0	0	0	0	0	0	0
2	0	∞	0	0	0	0	0	0	0
3	0	0	∞	0	0	0	0	0	0
4	0	0	0	∞	0	0	0	0	0
5	0	0	0	0	∞	0	0	0	0
6	0	0	0	0	0	∞	0	0	0
7	0	0	0	0	0	0	∞	0	0
8	0	0	0	0	0	0	0	∞	0
9	0	0	0	0	0	0	0	0	∞

one learned rule: ∞ costs on the diagonal

Figure: (Defresne et al. 2026a), Fig. 3

Raw inputs and missing data: visual Sudoku



clues = MNIST images

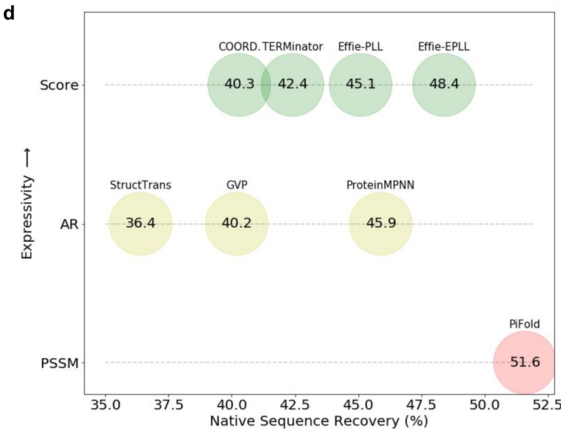
- Learning *at the same time* to read the digits and to play
- "Ungrounded" version: no label at all for the digits (no data leakage)

Approach	MNIST acc.	Perception	Solved	Time (h)
Rec. Trans.	99.4 %	74.8 %	59.0–75.6 %	5.1
NeSy Prog.	99.6–99.7 %	90.7–93.1 %	92.2–94.4 %	4.7
E-PLL	98.8 %	69.3–72.9 %	**92.1–93.4** %	**3.2**

Worse perception, yet the solver *fixes* 20 % of the misread grids.

Figure: (Defresne et al. 2026a), Tab. 3

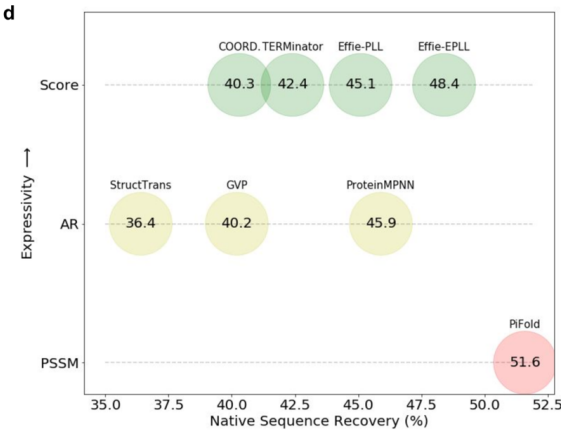
EffieDes recovers natural sequences better



- NSR (*native sequence recovery*): % of positions identical to the natural sequence
- Effie (E-PLL): 48.4 %; ProteinMPNN: 45.9 %
- Small proteins: Effie 33.0 % against Rosetta 17.9 %
- PiFold (51.6 %) predicts position by position: no constraints possible

Figure: (Defresne et al. 2026b), Fig. 2d

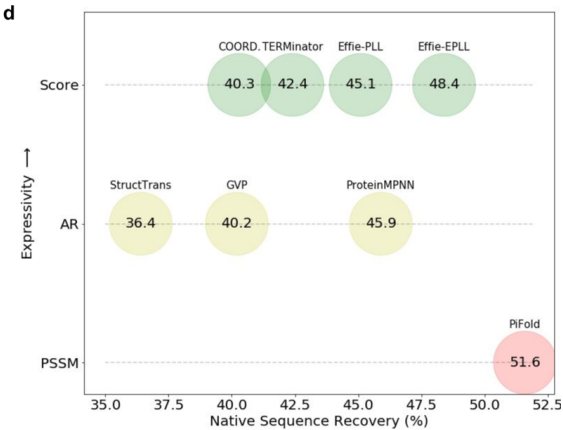
EffieDes recovers natural sequences better



- NSR (*native sequence recovery*): % of positions identical to the natural sequence
- Effie (E-PLL): 48.4 %; ProteinMPNN: 45.9 %
- Small proteins: Effie 33.0 % against Rosetta 17.9 %
- PiFold (51.6 %) predicts position by position: no constraints possible

Figure: (Defresne et al. 2026b), Fig. 2d

EffieDes recovers natural sequences better

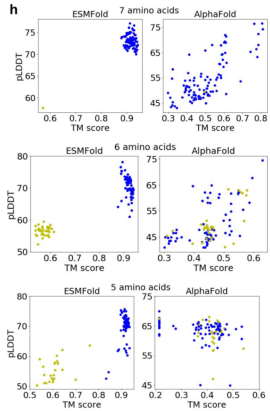


- NSR (*native sequence recovery*): % of positions identical to the natural sequence
- Effie (E-PLL): 48.4 %; ProteinMPNN: 45.9 %
- Small proteins: Effie 33.0 % against Rosetta 17.9 %
- PiFold (51.6 %) predicts position by position: no constraints possible

Figure: (Defresne et al. 2026b), Fig. 2d

Adding constraints without retraining: 5 to 7 amino acids

- DPBB barrel (core of RNA polymerase), symmetric
- Constraint: at most $k \in \{7, 6, 5\}$ amino acid types (unconstrained optimum: 12)
- No known natural protein is like this: no data to retrain on
- ESMFold: TM-score close to 1 for $k = 7$

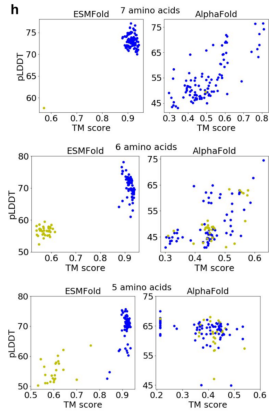


pLDDT (confidence) vs TM-score (agreement with the target)

Figure: (Defresne et al. 2026b), Fig. 2h (panels restacked); (Yagi et al. 2021)

Adding constraints without retraining: 5 to 7 amino acids

- DPBB barrel (core of RNA polymerase), symmetric
- Constraint: at most $k \in \{7, 6, 5\}$ amino acid types (unconstrained optimum: 12)
- No known natural protein is like this: no data to retrain on
- ESMFold: TM-score close to 1 for $k = 7$

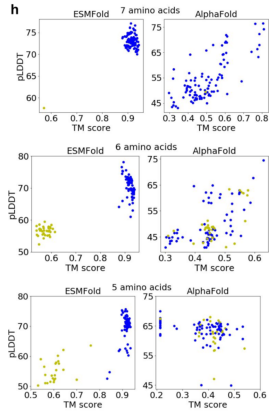


pLDDT (confidence) vs TM-score (agreement with the target)

Figure: (Defresne et al. 2026b), Fig. 2h (panels restacked); (Yagi et al. 2021)

Adding constraints without retraining: 5 to 7 amino acids

- DPBB barrel (core of RNA polymerase), symmetric
- Constraint: at most $k \in \{7, 6, 5\}$ amino acid types (unconstrained optimum: 12)
- No known natural protein is like this: no data to retrain on
- ESMFold: TM-score close to 1 for $k = 7$



pLDDT (confidence) vs TM-score (agreement with the target)

Figure: (Defresne et al. 2026b), Fig. 2h (panels restacked); (Yagi et al. 2021)

Demo — these constraints, live, in a few seconds

```
# a Boolean variable "this type is used"
for aa in range(20):
    problem.AddVariable(f"used_{aa}", [0, 1])
    for i in range(nb_var):    # xi = aa  =>  used = 1
        problem.AddFunction([i, nb_var + aa], pairs)

# at most max_type different types
problem.AddGeneralizedLinearConstraint(
    [(nb_var + aa, 1, 1) for aa in range(20)],
    "<=", max_type)
```

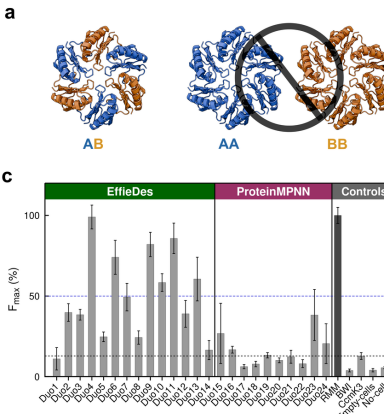
Extract from `Design-RNAP.py` (EffieDes repository), plus the order-2 symmetry.

max k	optimal score	types	solutions
20	-434.95	11	1
7	-423.92	7	1,401
6	-418.63	6	38
5	-405.72	5	59

Each row: under 7 s, *proved* optimum, *exhaustive* enumeration within 3 units of *that row's* optimum.

Code and data: `github.com/effiedes/EffieDes` — (Defresne et al. 2026b)

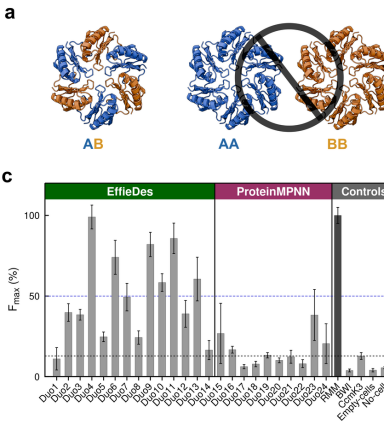
Designing *against*: pairs that assemble with each other



- Bacterial hexamer (BMC-H): obtain AB, avoid AA and BB (multi-state design)
- 100 % of EffieDes pairs favour AB (ProteinMPNN: ∼85 %)
- **Laboratory**: 12/14 EffieDes pairs assemble (86 %) against 2/10 for ProteinMPNN (20 %)
- 6/14 exceed 50 % of the natural control; none for ProteinMPNN

Figure: (Defresne et al. 2026b), Fig. 3a,c

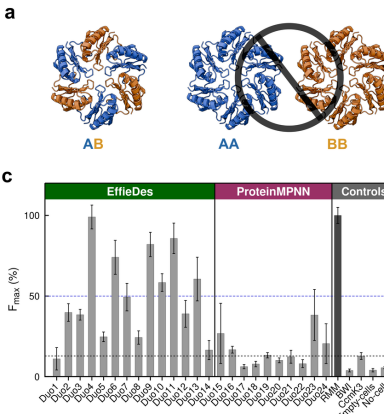
Designing *against*: pairs that assemble with each other



- Bacterial hexamer (BMC-H): obtain AB, avoid AA and BB (multi-state design)
- 100 % of EffieDes pairs favour AB (ProteinMPNN: ∼85 %)
- **Laboratory:** 12/14 EffieDes pairs assemble (86 %) against 2/10 for ProteinMPNN (20 %)
- 6/14 exceed 50 % of the natural control; none for ProteinMPNN

Figure: (Defresne et al. 2026b), Fig. 3a,c

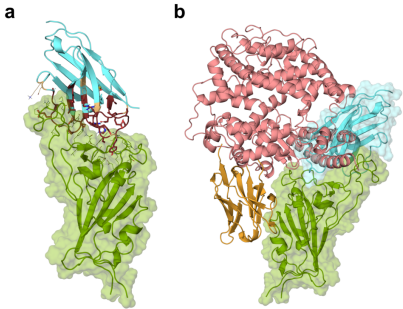
Designing *against*: pairs that assemble with each other



- Bacterial hexamer (BMC-H): obtain AB, avoid AA and BB (multi-state design)
- 100 % of EffieDes pairs favour AB (ProteinMPNN: ∼85 %)
- **Laboratory**: 12/14 EffieDes pairs assemble (86 %) against 2/10 for ProteinMPNN (20 %)
- 6/14 exceed 50 % of the natural control; none for ProteinMPNN

Figure: (Defresne et al. 2026b), Fig. 3a,c

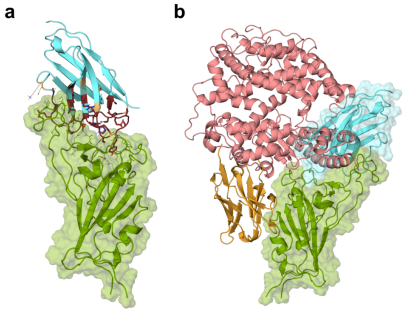
A single-domain antibody designed on a new backbone



- Target: a viral surface domain that the starting antibody no longer recognises
- Loop backbones generated by diffusion, then sequences designed
- Of 9 candidates tested, the only functional one comes from EffieDes
- Measured affinity: $K_D = 64$ nM, selective for the intended target

Figure: (Defresne et al. 2026b), Fig. 4a-b

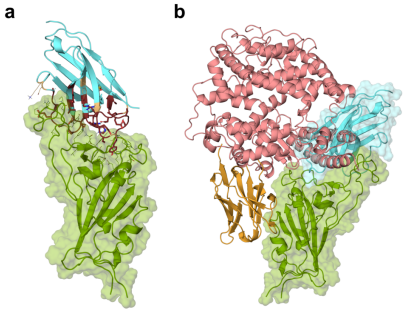
A single-domain antibody designed on a new backbone



- Target: a viral surface domain that the starting antibody no longer recognises
- Loop backbones generated by diffusion, then sequences designed
- Of 9 candidates tested, the only functional one comes from EffieDes
- Measured affinity: $K_D = 64$ nM, selective for the intended target

Figure: (Defresne et al. 2026b), Fig. 4a-b

A single-domain antibody designed on a new backbone



- Target: a viral surface domain that the starting antibody no longer recognises
- Loop backbones generated by diffusion, then sequences designed
- Of 9 candidates tested, the only functional one comes from EffieDes
- Measured affinity: $K_D = 64$ nM, selective for the intended target

Figure: (Defresne et al. 2026b), Fig. 4a-b

What exactness brings to the biologist

Without guarantees

- "here are 5,000 sampled sequences"
- constraints satisfied *on average*
- to change objective: retrain
- diversity = re-run the sampling

With an exact solver

- "here is the best sequence, and it is proved"
- constraints *always* satisfied
- a new objective: one line of model
- *guaranteed* diversity: every sequence at least k mutations from the others

The learned model remains inspectable: scores can be read, corrected, combined.

(Ruffini et al. 2021), (Defresne et al. 2026b)

From symbolic to neural, and the LLMs

Symbolic → neural

- Logical knowledge becomes a loss term (*semantic loss*)
- DeepProbLog, NeurASP: the network perceives, the logic program reasons
- Effie too: give it the constraints and it uses them to train the network
- Useful when the rules are *known* and the data scarce

LLM + solver

- The language model *translates* the statement into a formal model; the solver decides (Logic-LM)
- Or it *proposes* programs, which are evaluated and selected (FunSearch)
- LLMs alone fail on Sudoku, pathfinding, planning

As with Effie, the same division of labour: perceive and propose / verify and optimise.

(Manhaeve et al. 2018), (Yang et al. 2020), (Pan et al. 2023), (Romera-Paredes et al. 2024)

From symbolic to neural, and the LLMs

Symbolic → neural

- Logical knowledge becomes a loss term (*semantic loss*)
- DeepProbLog, NeurASP: the network perceives, the logic program reasons
- **Effie** too: give it the constraints and it uses them to train the network
- Useful when the rules are *known* and the data scarce

LLM + solver

- The language model *translates* the statement into a formal model; the solver decides (Logic-LM)
- Or it *proposes* programs, which are evaluated and selected (FunSearch)
- LLMs alone fail on Sudoku, pathfinding, planning

As with Effie, the same division of labour: perceive and propose / verify and optimise.

(Manhaeve et al. 2018), (Yang et al. 2020), (Pan et al. 2023), (Romera-Paredes et al. 2024)

From symbolic to neural, and the LLMs

Symbolic → neural

- Logical knowledge becomes a loss term (*semantic loss*)
- DeepProbLog, NeurASP: the network perceives, the logic program reasons
- **Effie** too: give it the constraints and it uses them to train the network
- Useful when the rules are *known* and the data scarce

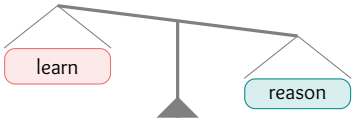
LLM + solver

- The language model *translates* the statement into a formal model; the solver decides (Logic-LM)
- Or it *proposes* programs, which are evaluated and selected (FunSearch)
- LLMs alone fail on Sudoku, pathfinding, planning

As with Effie, the same division of labour: perceive and propose / verify and optimise.

(Manhaeve et al. 2018), (Yang et al. 2020), (Pan et al. 2023), (Romera-Paredes et al. 2024)

G. Why believe a solver's answer?

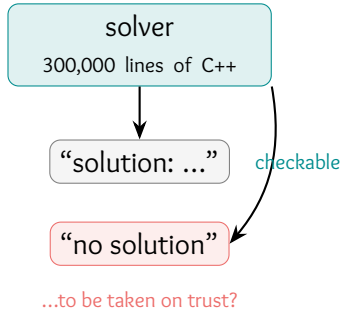


Can we learn and guarantee?

- A solver is a large piece of software
- Proof certificates
- A theorem proved by computer
- Lean, and the LLMs that prove

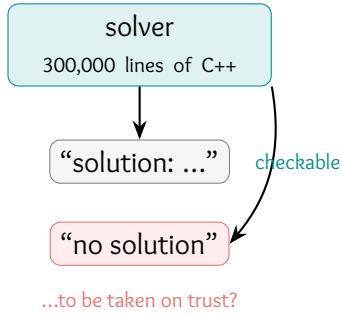
The paradox: trusting a very complex piece of software

- A SAT solver: hundreds of thousands of highly optimised lines
- "Satisfiable" is easy to check: just test the solution
- But "*un*satisfiable", or "this is the optimum"? Nothing to show…
- A bug there would be silent



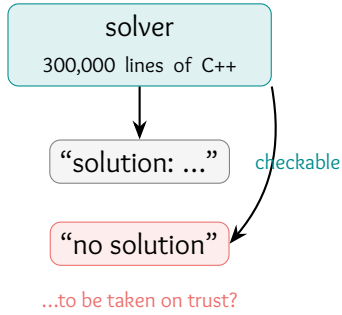
The paradox: trusting a very complex piece of software

- A SAT solver: hundreds of thousands of highly optimised lines
- "Satisfiable" is easy to check: just test the solution
- But "*un*satisfiable", or "this is the optimum"? Nothing to show…
- A bug there would be silent



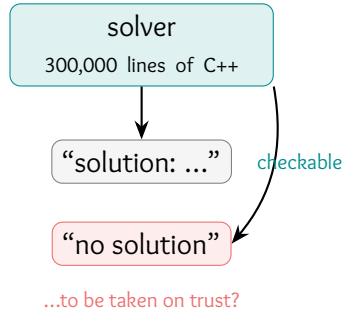
The paradox: trusting a very complex piece of software

- A SAT solver: hundreds of thousands of highly optimised lines
- "Satisfiable" is easy to check: just test the solution
- But "*un*satisfiable", or "this is the optimum"? Nothing to show...
- A bug there would be silent



The paradox: trusting a very complex piece of software

- A SAT solver: hundreds of thousands of highly optimised lines
- "Satisfiable" is easy to check: just test the solution
- But "*un*satisfiable", or "this is the optimum"? Nothing to show…
- A bug there would be silent



The answer: demand a certificate



- The DRAT format for SAT: the sequence of learned clauses, each one checkable
- We no longer trust the solver, but a checker of a few thousand lines
- Some checkers are themselves *proved correct* (in Coq, in ACL2)
- Standard in SAT competitions: no certificate, no points

(Cruz-Filipe et al. 2019)

The answer: demand a certificate



- The DRAT format for SAT: the sequence of learned clauses, each one checkable
- We no longer trust the solver, but a checker of a few thousand lines
- Some checkers are themselves *proved correct* (in Coq, in ACL2)
- Standard in SAT competitions: no certificate, no points

(Cruz-Filipe et al. 2019)

The answer: demand a certificate



- The DRAT format for SAT: the sequence of learned clauses, each one checkable
- We no longer trust the solver, but a checker of a few thousand lines
- Some checkers are themselves *proved correct* (in Coq, in ACL2)
- Standard in SAT competitions: no certificate, no points

(Cruz-Filipe et al. 2019)

The answer: demand a certificate

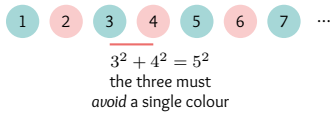


- The DRAT format for SAT: the sequence of learned clauses, each one checkable
- We no longer trust the solver, but a checker of a few thousand lines
- Some checkers are themselves *proved correct* (in Coq, in ACL2)
- Standard in SAT competitions: no certificate, no points

(Cruz-Filipe et al. 2019)

A theorem whose proof weighs 200 terabytes

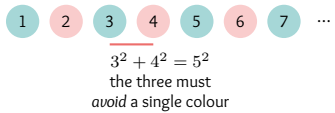
- Question (Graham): can the integers be coloured with two colours so that no triple $a^2 + b^2 = c^2$ is monochromatic?
- **Answer**: yes up to 7,824, no from 7,825 on
- SAT, 800 cores, 2 days; a DRAT certificate of $\sim$200 TB
- Then checked by a *certified* checker



(Heule et al. 2016), (Cruz-Filipe et al. 2019)

A theorem whose proof weighs 200 terabytes

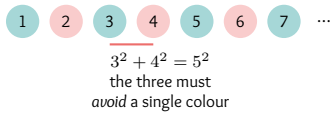
- Question (Graham): can the integers be coloured with two colours so that no triple $a^2 + b^2 = c^2$ is monochromatic?
- **Answer**: yes up to 7,824, no from 7,825 on
- SAT, 800 cores, 2 days; a DRAT certificate of ∼200 TB
- Then checked by a *certified* checker



(Heule et al. 2016), (Cruz-Filipe et al. 2019)

A theorem whose proof weighs 200 terabytes

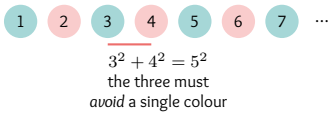
- Question (Graham): can the integers be coloured with two colours so that no triple $a^2 + b^2 = c^2$ is monochromatic?
- **Answer**: yes up to 7,824, no from 7,825 on
- SAT, 800 cores, 2 days; a DRAT certificate of $\sim$200 TB
- Then checked by a *certified* checker



(Heule et al. 2016), (Cruz-Filipe et al. 2019)

A theorem whose proof weighs 200 terabytes

- Question (Graham): can the integers be coloured with two colours so that no triple $a^2 + b^2 = c^2$ is monochromatic?
- **Answer**: yes up to 7,824, no from 7,825 on
- SAT, 800 cores, 2 days; a DRAT certificate of $\sim$200 TB
- Then checked by a *certified* checker



(Heule et al. 2016), (Cruz-Filipe et al. 2019)

Lean: a small trusted kernel

- A proof assistant: the human writes the proof, the machine checks it
- Every proof is reduced to a tiny *kernel* that a human can review
- Mathlib: hundreds of thousands of formalised theorems
- Also used to verify software

```
theorem add_comm' (a b : Nat) :
    a + b = b + a := by
  induction b with
  | zero => simp
  | succ n ih => omega
```

(Moura et al. 2021)

Lean: a small trusted kernel

- A proof assistant: the human writes the proof, the machine checks it
- Every proof is reduced to a tiny *kernel* that a human can review
- Mathlib: hundreds of thousands of formalised theorems
- Also used to verify software

```
theorem add_comm' (a b : Nat) :
    a + b = b + a := by
  induction b with
  | zero => simp
  | succ n ih => omega
```

(Moura et al. 2021)

Lean: a small trusted kernel

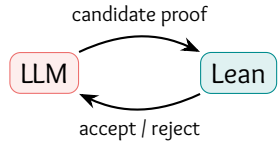
- A proof assistant: the human writes the proof, the machine checks it
- Every proof is reduced to a tiny *kernel* that a human can review
- Mathlib: hundreds of thousands of formalised theorems
- Also used to verify software

```
theorem add_comm' (a b : Nat) :
    a + b = b + a := by
  induction b with
  | zero => simp
  | succ n ih => omega
```

(Moura et al. 2021)

LLMs and proof assistants: propose, then verify

- An LLM writes candidate proofs in Lean; Lean accepts or rejects
- Rejection is a reliable, automatic, free learning signal
- AlphaProof: silver-medal level at the 2024 International Mathematical Olympiad

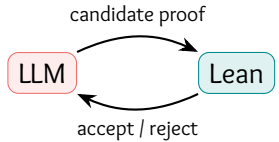


the LLM is allowed to be wrong: the checker does not lie

(Hubert et al. 2025)

LLMs and proof assistants: propose, then verify

- An LLM writes candidate proofs in Lean; Lean accepts or rejects
- Rejection is a reliable, automatic, free learning signal
- AlphaProof: silver-medal level at the 2024 International Mathematical Olympiad

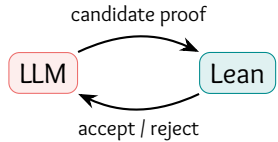


the LLM is allowed to be wrong:
the checker does not lie

(Hubert et al. 2025)

LLMs and proof assistants: propose, then verify

- An LLM writes candidate proofs in Lean; Lean accepts or rejects
- Rejection is a reliable, automatic, free learning signal
- AlphaProof: silver-medal level at the 2024 International Mathematical Olympiad



the LLM is allowed to be wrong:
the checker does not lie

(Hubert et al. 2025)

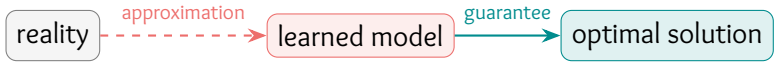
What the guarantees cover — and what they do not

Guaranteed

- The solution is optimal *for the model*
- The constraints posted are satisfied, all of them, always
- “There is no solution” is a certainty

Not guaranteed

- That the model describes reality
- That the network learned well (biased data, out of distribution)
- That the protein really folds: only the bench can say



An explicit model also means reproducibility

- The model (a CFN, an ASP program, an SMT formula) is a *file*: shareable, versionable, readable
- One instance, two different solvers, the same optimum: cross-checking
- The certificate (part G) makes the answer checkable by a third party
- Compare with: “I sampled 5,000 sequences with model X, version Y, seed Z”

Code and data for Effie and EffieDes: public repositories given in the papers.

(Defresne et al. 2026a), (Defresne et al. 2026b)

An explicit model also means reproducibility

- The model (a CFN, an ASP program, an SMT formula) is a *file*: shareable, versionable, readable
- One instance, two different solvers, the same optimum: cross-checking
- The certificate (part G) makes the answer checkable by a third party
- Compare with: “I sampled 5,000 sequences with model X, version Y, seed Z”

Code and data for Effie and EffieDes: public repositories given in the papers.

(Defresne et al. 2026a), (Defresne et al. 2026b)

An explicit model also means reproducibility

- The model (a CFN, an ASP program, an SMT formula) is a *file*: shareable, versionable, readable
- One instance, two different solvers, the same optimum: cross-checking
- The certificate (part G) makes the answer checkable by a third party
- Compare with: “I sampled 5,000 sequences with model X, version Y, seed Z”

Code and data for Effie and EffieDes: public repositories given in the papers.

(Defresne et al. 2026a), (Defresne et al. 2026b)

An explicit model also means reproducibility

- The model (a CFN, an ASP program, an SMT formula) is a *file*: shareable, versionable, readable
- One instance, two different solvers, the same optimum: cross-checking
- The certificate (part G) makes the answer checkable by a third party
- Compare with: “I sampled 5,000 sequences with model X, version Y, seed Z”

Code and data for Effie and EffieDes: public repositories given in the papers.

(Defresne et al. 2026a), (Defresne et al. 2026b)

Two bridges: towards ethics, and towards knowledge graphs

Ethics day

- An explicit constraint is *auditable*: it can be read, discussed, contested
- A preference buried in weights is not
- But: who chooses the constraints? And what is not modelled is not protected

Tomorrow: knowledge graphs

- Same principle: explicit representation + inference
- Description logics: decidable fragments, dedicated reasoners
- A shared question: where does the knowledge come from? Experts, or learned?

The “explicit knowledge / learned knowledge” tension runs through both days.

Three take-home messages

1 **Model, then solve is a method, not an old memory.**
Many research problems can be written in a few dozen declarative lines, and a generic solver handles them.

2 **NP-hard is not a refusal.**
It is a worst-case property. Try a solver on your instances before concluding.

3 **Neuro-symbolic AI is a division of labour.**
The network perceives and proposes; the solver verifies and optimises. We keep learning from raw data *and* the guarantees.

Going further

Handbooks

- *Handbook of Satisfiability*, 2nd ed.
- *Handbook of Constraint Programming*
- Cooper et al., *Graphical Models* (discrete graphical models)
- Baader et al., *Introduction to Description Logic*

Neuro-symbolic surveys

- Kautz 2022; Garcez & Lamb 2023
- Marra et al. 2024; Bhuyan et al. 2024

Tools (open source)

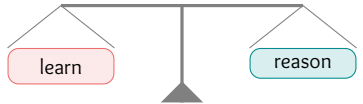
- `toulbar2` — weighted constraints, CFN
- `Z3`, `cvc5` — SMT
- `clingo` — ASP
- `OR-Tools`, `MiniZinc` — constraints
- `HiGHS`, `SCIP` — MILP
- `Lean` + Mathlib — proof
- `ProbLog`, `Scallop`, `NeurASP` — neuro-symbolic
- `PySDD`, `PyJuice` — compilation, circuits

This course

- Backup slides: a separate PDF

(Biere et al. 2021), (Rossi et al. 2006), (Cooper et al. 2020), (Baader et al. 2017)

Thank you — questions



Learn *and* guarantee:
the network perceives and proposes,
the solver verifies and optimises.

thomas.schiex@inrae.fr — INRAE, MIAT & ANITI, Toulouse

Work presented: M. Defresne, D. Dessaux, R. Gambardella, S. Buchet, S. de Givry, V. Durante, S. Barbe
and the whole of the MIAT (INRAE) and TBI (INSA Toulouse) teams

References I

Allouche, D. et al. (2014). “Computational protein design as an optimization problem”. In: *Artificial Intelligence* 212, pp. 59–79.

Allouche, D. et al. (2015). “Anytime hybrid best-first search with tree decomposition for weighted CSP”. In: *International Conference on Principles and Practice of Constraint Programming*. Springer, pp. 12–29.

Arora, S. and B. Barak (2009). *Computational Complexity: A Modern Approach*. Cambridge University Press. DOI: 10.1017/CBO9780511804090.

Baader, F., I. Horrocks, C. Lutz, and U. Sattler (2017). *An Introduction to Description Logic*. Cambridge University Press. DOI: 10.1017/9781139025355.

Barbosa, H. et al. (2022). “cvc5: A Versatile and Industrial-Strength SMT Solver”. In: *TACAS 2022*. Vol. 13243. LNCS, pp. 415–442. DOI: 10.1007/978-3-030-99524-9_24.

Bekkum, M. van et al. (2021). “Modular design patterns for hybrid learning and reasoning systems”. In: *Applied Intelligence* 51.9, pp. 6528–6546. DOI: 10.1007/s10489-021-02394-3.

Bertelè, U. and F. Brioschi (1972). *Nonserial Dynamic Programming*. Vol. 91. Mathematics in Science and Engineering. New York: Academic Press.

Besag, J. (1975). “Statistical analysis of non-lattice data”. In: *Journal of the Royal Statistical Society: Series D (The Statistician)* 24.3, pp. 179–195.

Bhuyan, B. P., A. Ramdane-Cherif, R. Tomar, and T. P. Singh (2024). “Neuro-symbolic artificial intelligence: a survey”. In: *Neural Computing and Applications* 36.21, pp. 12809–12844. DOI: 10.1007/s00521-024-09960-z.

References II

Biere, A., M. Heule, H. van Maaren, and T. Walsh, eds. (2021). *Handbook of Satisfiability (2nd edition)*. Vol. 336. Frontiers in Artificial Intelligence and Applications. IOS Press. DOI: 10.3233/FAIA336.

Brouard, C., S. de Givry, and T. Schiex (2020). "Pushing data into CP models using graphical model learning and solving". In: *International Conference on Principles and Practice of Constraint Programming*. Springer, pp. 811–827.

Chang, O., L. Flokas, H. Lipson, and M. Spranger (2020). "Assessing SATNet's Ability to Solve the Symbol Grounding Problem". In: *Advances in Neural Information Processing Systems*. Ed. by H. Larochelle et al. Vol. 33. Curran Associates, Inc., pp. 1428–1439. URL: https://proceedings.neurips.cc/paper/2020/file/0ff8033cf9437c213ee13937b1c4c455-Paper.pdf.

Cook, S. A. (1971). "The complexity of theorem-proving procedures". In: *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing (STOC)*, pp. 151–158. DOI: 10.1145/800157.805047.

Cooper, M., S. de Givry, and T. Schiex (2020). "Graphical models: queries, complexity, algorithms". In: *Symposium on Theoretical Aspects of Computer Science, Leibniz International Proceedings in Informatics*. Vol. 154, pp. 4–1.

Cruz-Filipe, L., J. Marques-Silva, and P. Schneider-Kamp (2019). "Formally Verifying the Solution to the Boolean Pythagorean Triples Problem". In: *Journal of Automated Reasoning* 63, pp. 695–722. DOI: 10.1007/s10817-018-9490-4.

Darwiche, A. (2003). "A differential approach to inference in Bayesian networks". In: *Journal of the ACM* 50.3, pp. 280–305. DOI: 10.1145/765568.765570.

Darwiche, A. and P. Marquis (2002). "A Knowledge Compilation Map". In: *Journal of Artificial Intelligence Research* 17, pp. 229–264.

References III

Dechter, R. and I. Rish (1994). “Directional Resolution: The Davis–Putnam Procedure, Revisited”. In: *Principles of Knowledge Representation and Reasoning (KR-94)*, pp. 134–145.

Defresne, M., S. Barbe, and T. Schiex (2023). “Scalable Coupling of Deep Learning with Logical Reasoning”. In: *Proceedings of the 32nd International Joint Conference on Artificial Intelligence, IJCAI 2023*, pp. 3615–3623. DOI: 10.24963/IJCAI.2023/402.

Defresne, M., R. Gambardella, S. Barbe, and T. Schiex (2026a). “Scaling Neuro-symbolic Problem Solving: Solver-Free Learning of Constraints and Objectives”. In: *Journal of Artificial Intelligence Research* 84, Article 3. DOI: 10.1613/jair.1.21105.

Defresne, M. et al. (2026b). “A Generative Neuro-Symbolic AI for Protein Sequence Design”. In: *Advanced Science*. DOI: 10.1002/advs.76464.

DeLong, L. N., R. F. Mir, and J. D. Fleuriot (2024). “Neurosymbolic AI for Reasoning Over Knowledge Graphs: A Survey”. In: *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–21. DOI: 10.1109/TNNLS.2024.3420218.

Dubois, D., J. Lang, and H. Prade (1994). “Possibilistic Logic”. In: *Handbook of Logic in Artificial Intelligence and Logic Programming, Volume 3: Nonmonotonic Reasoning and Uncertain Reasoning*. Ed. by D. M. Gabbay, C. J. Hogger, and J. A. Robinson. Oxford University Press, pp. 439–513.

Durante, V., G. Katsirelos, and T. Schiex (2022). “Efficient low rank convex bounds for pairwise discrete Graphical Models”. In: *39th International Conference on Machine Learning*.

Garcez, A. d’Avila and L. C. Lamb (2023). “Neurosymbolic AI: the 3rd wave”. In: *Artificial Intelligence Review* 56.11, pp. 12387–12406. DOI: 10.1007/s10462-023-10448-w.

References IV

Gebser, M., R. Kaminski, B. Kaufmann, and T. Schaub (2019). “Multi-shot ASP solving with clingo”. In: *Theory and Practice of Logic Programming* 19.1, pp. 27–82. DOI: 10.1017/S1471068418000054.

Heule, M. J. H., O. Kullmann, and V. W. Marek (2016). “Solving and Verifying the Boolean Pythagorean Triples Problem via Cube-and-Conquer”. In: *Theory and Applications of Satisfiability Testing – SAT 2016*. Vol. 9710. LNCS, pp. 228–245. DOI: 10.1007/978-3-319-40970-2_15.

Hubert, T., R. Mehta, L. Sartran, et al. (2025). “Olympiad-level formal mathematical reasoning with reinforcement learning”. In: *Nature* 651, pp. 607–613. DOI: 10.1038/s41586-025-09833-y.

Hurley, B. et al. (2016). “Multi-language evaluation of exact solvers in graphical model discrete optimization”. In: *Constraints* 21, pp. 413–434.

Ingraham, J., V. K. Garg, R. Barzilay, and T. Jaakkola (2019). “Generative models for graph-based protein design”. In: *33rd Conference on Neural Information Processing Systems (NeurIPS 2019)*.

Kautz, H. A. (2022). “The Third AI Summer: AAAI Robert S. Engelmore Memorial Lecture”. In: *AI Magazine* 43.1, pp. 105–125. DOI: 10.1002/aaai.12036.

Koller, D. and N. Friedman (2009). *Probabilistic graphical models: principles and techniques*. MIT press.

Li, T. et al. (July 2021). “A synthetic nanobody targeting RBD protects hamsters from SARS-CoV-2 infection”. In: *Nature Communications* 12.1, p. 4635. ISSN: 2041-1723. DOI: 10.1038/s41467-021-24905-z.

Li, Z. et al. (2023). “Neuro-symbolic learning yielding logical constraints”. In: *Advances in Neural Information Processing Systems* 36, pp. 21635–21657.

References V

Manhaeve, R. et al. (2018). “DeepProbLog: Neural probabilistic logic programming”. In: *Advances in neural information processing systems* 31.

Marques-Silva, J. P. and K. A. Sakallah (1999). “GRASP: a search algorithm for propositional satisfiability”. In: *IEEE Transactions on Computers* 48.5, pp. 506–521. DOI: 10.1109/12.769433.

Marra, G., S. Dumančić, R. Manhaeve, and L. De Raedt (2024). “From statistical relational to neurosymbolic artificial intelligence: A survey”. In: *Artificial Intelligence* 328, p. 104062. DOI: 10.1016/j.artint.2023.104062.

Montalbano, P., S. de Givry, and G. Katsirelos (2022). “Multiple-choice knapsack constraint in graphical models”. In: *Integration of Constraint Programming, Artificial Intelligence, and Operations Research*. Ed. by P. Schaus. Cham: Springer International Publishing, pp. 282–299. ISBN: 978-3-031-08011-1. DOI: 10.1007/978-3-031-08011-1_19.

Montanari, A. and J. Pereira (2009). “Which graphical models are difficult to learn?” In: *Advances in Neural Information Processing Systems*. Ed. by Y. Bengio et al. Vol. 22. Curran Associates, Inc. URL: https://proceedings.neurips.cc/paper/2009/file/22fb0cee7e1f3bde58293de743871417-Paper.pdf.

Moura, L. de and N. Bjørner (2008). “Z3: An Efficient SMT Solver”. In: *TACAS 2008*. Vol. 4963. LNCS, pp. 337–340. DOI: 10.1007/978-3-540-78800-3_24.

Moura, L. de and S. Ullrich (2021). “The Lean 4 Theorem Prover and Programming Language”. In: *Automated Deduction – CADE 28*. Vol. 12699. LNCS, pp. 625–635. DOI: 10.1007/978-3-030-79876-5_37.

Palm, R., U. Paquet, and O. Winther (2018). “Recurrent Relational Networks”. In: *Advances in Neural Information Processing Systems*. Ed. by S. Bengio et al. Vol. 31. Curran Associates, Inc. URL: https://proceedings.neurips.cc/paper/2018/file/b9f94c77652c9a76fc8a442748cd54bd-Paper.pdf.

References VI

Pan, L., A. Albalak, X. Wang, and W. Y. Wang (2023). “Logic-LM: Empowering Large Language Models with Symbolic Solvers for Faithful Logical Reasoning”. In: *Findings of the Association for Computational Linguistics: EMNLP 2023*, pp. 3806–3824. DOI: 10.18653/v1/2023.findings-emnlp.248.

Pierce, N. A. and E. Winfree (2002). “Protein design is NP-hard”. In: *Protein engineering* 15.10, pp. 779–782.

Raissi, M., P. Perdikaris, and G. E. Karniadakis (2019). “Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations”. In: *Journal of Computational Physics* 378, pp. 686–707. DOI: 10.1016/j.jcp.2018.10.045.

Renkhoff, J. et al. (2024). “A Survey on Verification and Validation, Testing and Evaluations of Neurosymbolic Artificial Intelligence”. In: *IEEE Transactions on Artificial Intelligence* 5.8, pp. 3765–3779. DOI: 10.1109/TAI.2024.3351980.

Romera-Paredes, B., M. Barekatain, A. Novikov, M. Balog, et al. (2024). “Mathematical discoveries from program search with large language models”. In: *Nature* 625, pp. 468–475. DOI: 10.1038/s41586-023-06924-6.

Rossi, F., P. van Beek, and T. Walsh, eds. (2006). *Handbook of Constraint Programming*. Elsevier.

Ruffini, M. et al. (2021). “Guaranteed diversity and optimality in cost function network based computational protein design methods”. In: *Algorithms* 14.6, p. 168.

Schiex, T., H. Fargier, and G. Verfaillie (1995). “Valued Constraint Satisfaction Problems: Hard and Easy Problems”. In: *Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI-95)*. Montréal, Canada, pp. 631–637.

References VII

Shlomi, T., Y. Eisenberg, R. Sharan, and E. Ruppin (2007). “A genome-scale computational study of the interplay between transcriptional regulation and metabolism”. In: *Molecular Systems Biology* 3, p. 101. DOI: 10.1038/msb4100141.

Simoncini, D. et al. (2015). “Guaranteed Discrete Energy Optimization on Large Protein Design Problems”. In: *Journal of Chemical Theory and Computation* 11.12, pp. 5980–5989. DOI: 10.1021/acs.jctc.5b00594.

Wang, G. et al. (2025). “Hierarchical Reasoning Model”. In: *arXiv preprint arXiv:2506.21734*.

Wang, P.-W., P. Donti, B. Wilder, and Z. Kolter (June 2019). “SATNet: Bridging deep learning and logical reasoning using a differentiable satisfiability solver”. In: *Proceedings of the 36th International Conference on Machine Learning*. Ed. by K. Chaudhuri and R. Salakhutdinov. Vol. 97. Proceedings of Machine Learning Research. PMLR, pp. 6545–6554. URL: https://proceedings.mlr.press/v97/wang19e.html.

Watson, J. L. et al. (Aug. 2023). “De novo design of protein structure and function with RFdiffusion”. In: *Nature* 620.7976, pp. 1089–1100. ISSN: 1476-4687. DOI: 10.1038/s41586-023-06415-8.

Xu, J. et al. (July 2018). “A Semantic Loss Function for Deep Learning with Symbolic Knowledge”. In: *Proceedings of the 35th International Conference on Machine Learning*. Ed. by J. Dy and A. Krause. Vol. 80. Proceedings of Machine Learning Research. PMLR, pp. 5502–5511. URL: https://proceedings.mlr.press/v80/xu18h.html.

Yagi, S. et al. (2021). “Seven amino acid types suffice to create the core fold of RNA polymerase”. In: *Journal of the American Chemical Society* 143.39, pp. 15998–16006.

References VIII

Yang, Z., A. Ishay, and J. Lee (2020). "NeurASP: Embracing Neural Networks into Answer Set Programming". In: *29th International Joint Conference on Artificial Intelligence (IJCAI 2020)*. DOI: 10.24963/ijcai.2020/243. URL: https://par.nsf.gov/biblio/10184165.

— (2023). "Learning to Solve Constraint Satisfaction Problems with Recurrent Transformer". In: *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net. URL: https://openreview.net/forum?id=udNhDCr2KQe.

Yato, T. and T. Seta (2003). "Complexity and Completeness of Finding Another Solution and Its Application to Puzzles". In: *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences* E86-A.5, pp. 1052–1060.

Ye, J. et al. (2025). "Beyond Autoregression: Discrete Diffusion for Complex Reasoning and Planning". In: *The Thirteenth International Conference on Learning Representations*. URL: https://openreview.net/forum?id=NRYgUzSPZz.