Automatic Error Function Learning with Interpretable Compositional Networks

Florian Richoux
joint work with Jean-François Baffier, RIKEN AIP

June 25, 2020

Outline

Outline

What is Constraint Programming?

What is CP?

Set of methods to model and solve combinatorial problems.

Constraint Network

A constraint network (CN) is defined by a tuple (V, D, C) such that:

$$\text{CN} = \left[\begin{array}{ll} V: & \text{Set of variables.} \\ D: & \text{Domain (set of possible values of variables).} \\ C: & \text{Set of constraints (\textit{i.e.}, predicates).} \end{array}\right.$$

Constraint Satisfaction Problem (CSP)

Given a constraint network, does a solution exist?

Example: the 3-color problem

CN for 3-color

Variables $V = \{v_1, \ldots, v_n\}$, one variable for each vertex.

Domain $D = \{0, 1, 2\}$, one value for each color.

Constraint $\neq$, one per edge.

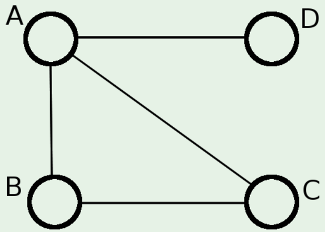
Example: the 3-color problem

CN for 3-color

Variables $V = \{v_1, \ldots, v_n\}$, one variable for each vertex.

Domain $D = \{0, 1, 2\}$, one value for each color.

Constraint $\neq$, one per edge.



CSP formula

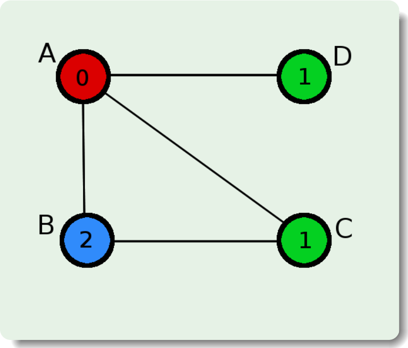
$(a \neq b) \wedge (a \neq c) \wedge$
$(b \neq c) \wedge (a \neq d)$

CN for 3-color

Variables $V = \{v_1, \ldots, v_n\}$, one variable for each vertex.

Domain $D = \{0, 1, 2\}$, one value for each color.

Constraint $\neq$, one per edge.



CSP formula

$(a \neq b) \wedge (a \neq c) \wedge$
$(b \neq c) \wedge (a \neq d)$

A solution

$a = 0$, $b = 2$, $c = d = 1$

Intuitive definition of EFN

Error Function Networks (EFN)

$$\text{EFN} = \left[\begin{array}{ll} V: & \text{Set of variables.} \\ D: & \text{Domain (set of possible values of variables).} \\ F: & \text{Set of error functions } f_c : D^k \to \mathbb{R}^+. \end{array}\right.$$

Intuitive definition of EFN

Error Function Networks (EFN)

$$\text{EFN} = \left[\begin{array}{ll} V: & \text{Set of variables.} \\ D: & \text{Domain (set of possible values of variables).} \\ F: & \text{Set of error functions } f_c : D^k \to \mathbb{R}^+. \end{array} \right.$$

Intuition behind error functions

Let (x_1, x_2, x_3) be an assignment for f_c:

- If $f_c(x_1, x_2, x_3) = 0$ then (x_1, x_2, x_3) **satisfies** the constraint c.
- If $f_c(x_1, x_2, x_3)$ is small then (x_1, x_2, x_3) is **close to** satisfy c.
- If $f_c(x_1, x_2, x_3)$ is high then (x_1, x_2, x_3) is **far from** satisfying c.

Intuitive definition of EFN

Error Function Networks (EFN)

$$\text{EFN} = \left[\begin{array}{ll} V: & \text{Set of variables.} \\ D: & \text{Domain (set of possible values of variables).} \\ F: & \text{Set of error functions } f_c : D^k \rightarrow \mathbb{R}^+. \end{array} \right.$$

Intuition behind error functions

Let (x_1, x_2, x_3) be an assignment for f_c:

- If $f_c(x_1, x_2, x_3) = 0$ then (x_1, x_2, x_3) **satisfies** the constraint c.
- If $f_c(x_1, x_2, x_3)$ is small then (x_1, x_2, x_3) is **close to** satisfy c.
- If $f_c(x_1, x_2, x_3)$ is high then (x_1, x_2, x_3) is **far from** satisfying c.

Error Function Satisfaction Problem (EFSP)

Given a error function network, does a solution exist?

Intuitive definition of EFN

Constraint representation

Error function = degree of dissatisfaction of a constraint.

For example

Consider $f_c(x, y) := |x - y|$ (representing the constraint $x = y$)

- With $x = 4$ and $y = 4$, $f_c(4, 4) = 0$
- With $x = 4$ and $y = 5$, $f_c(4, 5) = 1$
- With $x = 4$ and $y = 500$, $f_c(4, 500) = 496$

Outline

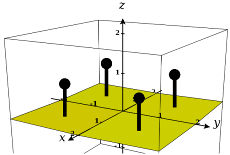
Why EFN?

Offers a landscape on assignments $\vec{x}$.

Offers a landscape on assignments $\vec{x}$.

No structures

$$l(\vec{x}) = \begin{cases} 1 & \text{if } \vec{x} \text{ is a solution} \\ 0 & \text{otherwise} \end{cases}$$

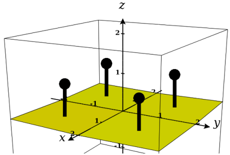


Why EFN?

Offers a landscape on assignments $\vec{x}$.

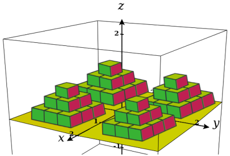
No structures

$$l(\vec{x}) = \begin{cases} 1 & \text{if } \vec{x} \text{ is a solution} \\ 0 & \text{otherwise} \end{cases}$$



Constraint Network

$l(\vec{x}) = \#\{\text{Number of satisfied constraints}\}$

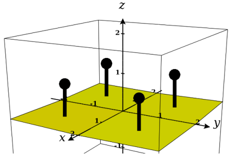


Why EFN?

Offers a landscape on assignments $\vec{x}$.

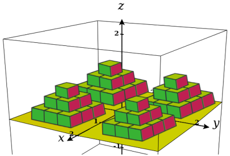
No structures

$$l(\vec{x}) = \begin{cases} 1 & \text{if } \vec{x} \text{ is a solution} \\ 0 & \text{otherwise} \end{cases}$$



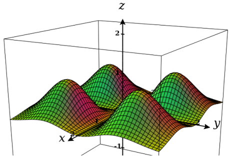
Constraint Network

$l(\vec{x}) = \#\{\text{Number of satisfied constraints}\}$



Error Function Network

$$l(\vec{x}) = \sum_{f_c \in F} f_c(\vec{x})$$



Why EFN?

Pros

Solvers can exploit efficiently this landscape.

Why EFN?

Pros

Solvers can exploit efficiently this landscape.

Cons

Making a EFN model is complicated: what is a good error function?

Why EFN?

Pros

Solvers can exploit efficiently this landscape.

Cons

Making a EFN model is complicated: what is a good error function?

For example

Is $f_c(x, y) = |x - y|$ relevant for the constraint $x = y$?

- If $x = 4$ and $y = 5$, then change y to 4 or x to 5 $\Rightarrow$ 1 action.
- If $x = 4$ and $y = 500$, then change y to 4 or x to 500 $\Rightarrow$ 1 action.

Why EFN?

Pros

Solvers can exploit efficiently this landscape.

Cons

Making a EFN model is complicated: what is a good error function?

For example

Is $f_c(x, y) = |x - y|$ relevant for the constraint $x = y$?

- If $x = 4$ and $y = 5$, then change y to 4 or x to 5 $\Rightarrow$ 1 action.
- If $x = 4$ and $y = 500$, then change y to 4 or x to 500 $\Rightarrow$ 1 action.

Outline

Learning Error Functions

Error functions seen as (non-linear) combination of elementary operations.

Goal

For each constraint, learn a good combination of elementary operations.

The user provides a CN

$$\left[\begin{array}{ll} V: & \text{Variables} \\ D: & \text{Domain} \\ C: & \text{Constraints} \end{array}\right.$$

The user gets an EFN

$$\left[\begin{array}{ll} V: & \text{Variables} \\ D: & \text{Domain} \\ \mathbf{F}: & \textbf{Error functions} \end{array}\right.$$

Learning Error Functions

Supervised learning

Learn error functions similar to the Hamming error.

Hamming error $h_c(\vec{x})$

$h_c(\vec{x})$: minimal number of values from $\vec{x}$ to change to get a solution.

Loss function of our supervised learning

Let θ_c be our model for one error function f_c.

$$\mathcal{L}(\theta_c, h_c) = \sum_{\vec{x}} |\theta_c(\vec{x}) - h_c(\vec{x})|$$

Learning Error Functions

Supervised learning

Learn error functions similar to the Hamming error.

Hamming error $h_c(\vec{x})$

$h_c(\vec{x})$: minimal number of values from $\vec{x}$ to change to get a solution.

Loss function of our supervised learning

Let θ_c be our model for one error function f_c.

$$\mathcal{L}(\theta_c, h_c) = \sum_{\vec{x}} |\theta_c(\vec{x}) - h_c(\vec{x})|$$

So what is our model θ_c?

Idea based upon CPPN

Our model is a variation of Compositional Pattern-Producing Networks.

Regular neural networks

Usually, neurons in NN contains sigmoid-like functions only, like ReLU.

Idea based upon CPPN

Our model is a variation of Compositional Pattern-Producing Networks.

Regular neural networks

Usually, neurons in NN contains sigmoid-like functions only, like ReLU.

CPPN idea

CPPN's neurons can contain many other kinds of functions:
sigmoids, Gaussians, trigonometric functions, linear functions, ...

Our model is a variation of Compositional Pattern-Producing Networks.

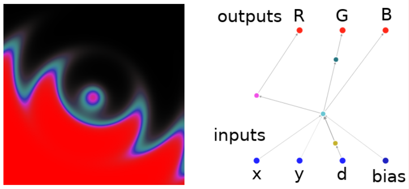
Regular neural networks

Usually, neurons in NN contains sigmoid-like functions only, like ReLU.

CPPN idea

CPPN's neurons can contain many other kinds of functions:
sigmoids, Gaussians, trigonometric functions, linear functions, ...

CPPN used to make 2D/3D images (source: otoro.net/neurogram/)



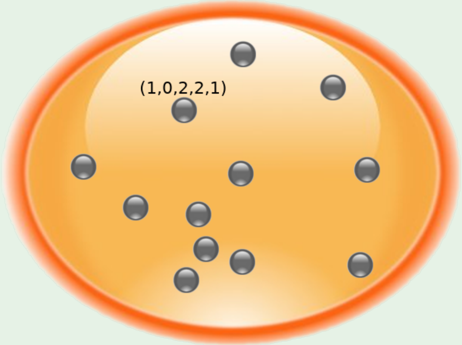
We take 2 ideas from CPPN to make ICN

- Neurons can contain one operation among many possible ones,
- ICN deals with an input space by taking one by one all elements.

We take 2 ideas from CPPN to make ICN

- Neurons can contain one operation among many possible ones,
- ICN deals with an input space by taking one by one all elements.

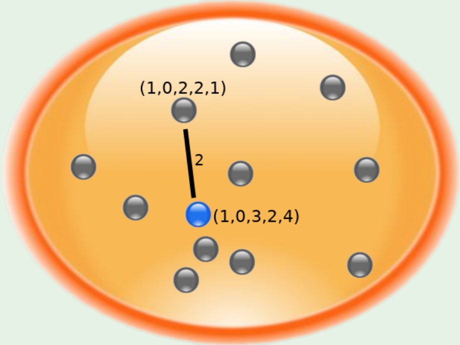
Input space of **one given constraint**



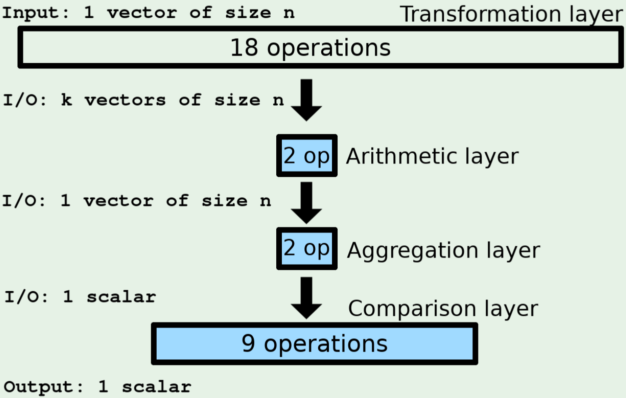
We take 2 ideas from CPPN to make ICN

- Neurons can contain one operation among many possible ones,
- ICN deals with an input space by taking one by one all elements.

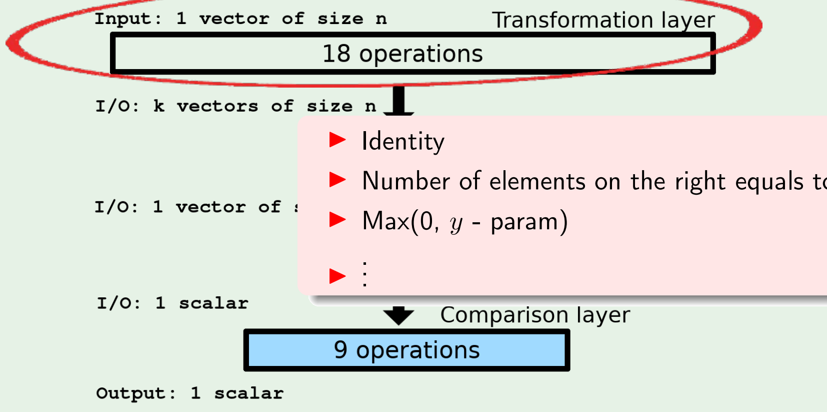
Input space of **one given constraint**



Our ICN architecture

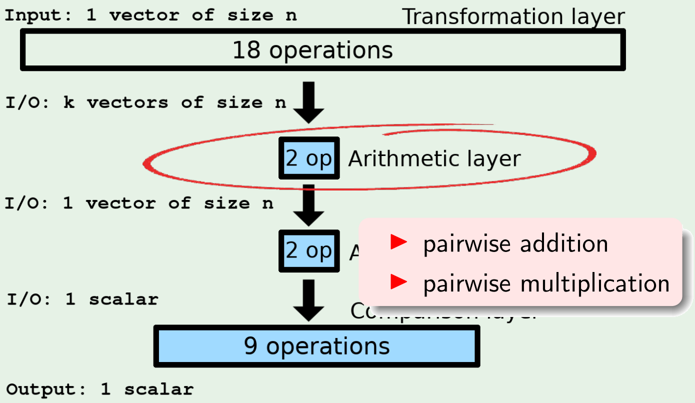


Our ICN architecture

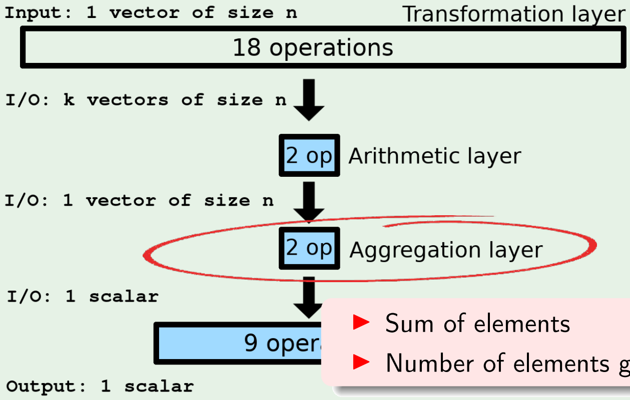


- Identity
- Number of elements on the right equals to y
- Max(0, y - param)
- $\vdots$

Our ICN architecture

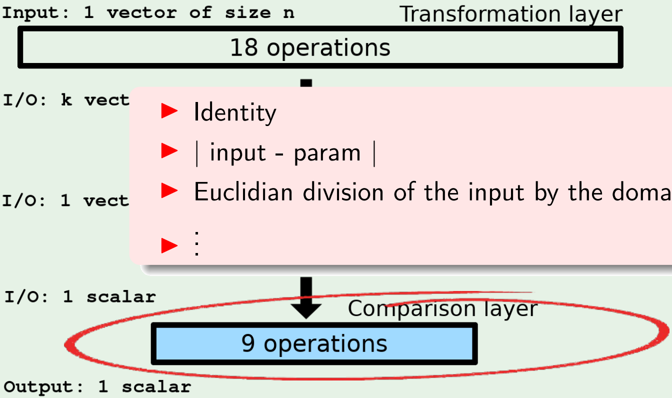


Our ICN architecture



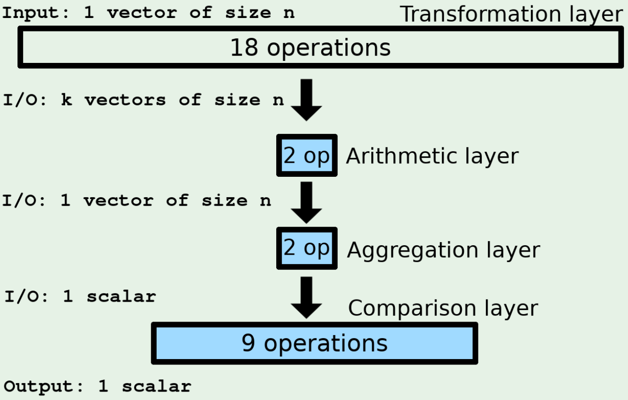
- Sum of elements
- Number of elements greater than 0

Our ICN architecture

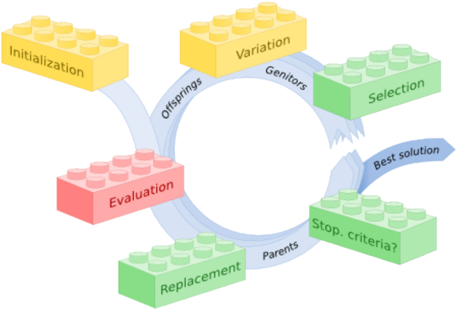


- Identity
- | input - param |
- Euclidian division of the input by the domain size
- ⋮

Our ICN architecture

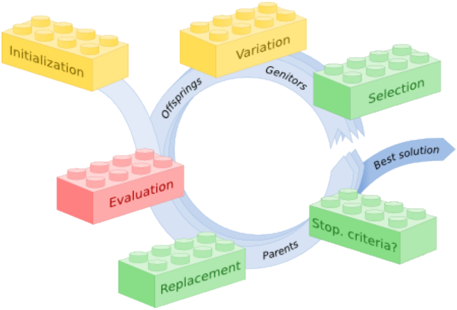


Learn ICN with Genetic Algorithms



Individual modeling

Vector of 29 bits:
one bit for each operations.



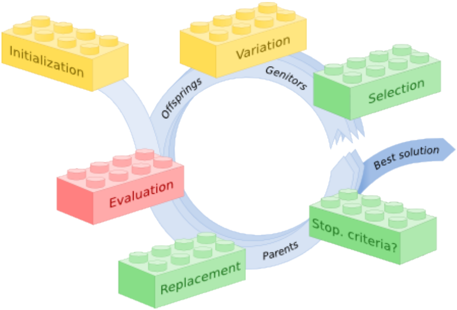
Learn ICN with Genetic Algorithms

Individual modeling

Vector of 29 bits:
one bit for each operations.

Initialization

Draw 100 individuals randomly.



Learn ICN with Genetic Algorithms

Individual modeling

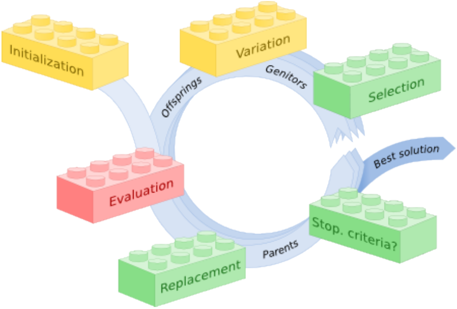
Vector of 29 bits:
one bit for each operations.

Initialization

Draw 100 individuals randomly.

Evaluation

Minimize the loss function $\mathcal{L}(\theta_c, h_c)$.



Learn ICN with Genetic Algorithms

Individual modeling

Vector of 29 bits:
one bit for each operations.

Initialization

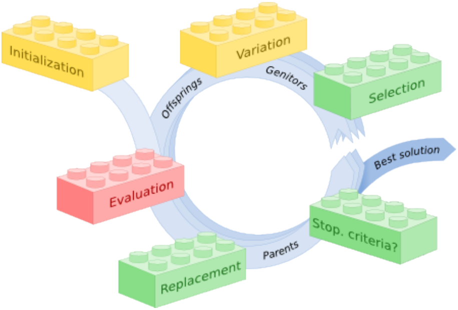
Draw 100 individuals randomly.

Evaluation

Minimize the loss function $\mathcal{L}(\theta_c, h_c)$.

Replacement

Elitism merge and deterministic tournament to keep 100 individuals.



Learn ICN with Genetic Algorithms

Individual modeling

Vector of 29 bits:
one bit for each operations.

Initialization

Draw 100 individuals randomly.

Evaluation

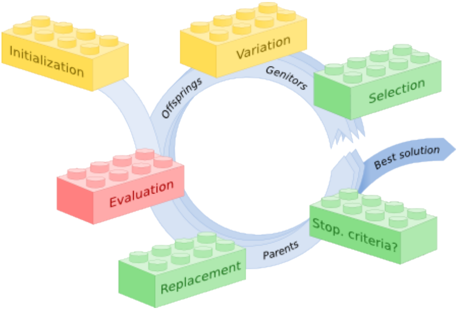
Minimize the loss function $\mathcal{L}(\theta_c, h_c)$.

Replacement

Elitism merge and deterministic tournament to keep 100 individuals.

Stop criteria

Reaching 400 generations.



Learn ICN with Genetic Algorithms

Individual modeling

Vector of 29 bits:
one bit for each operations.

Initialization

Draw 100 individuals randomly.

Evaluation

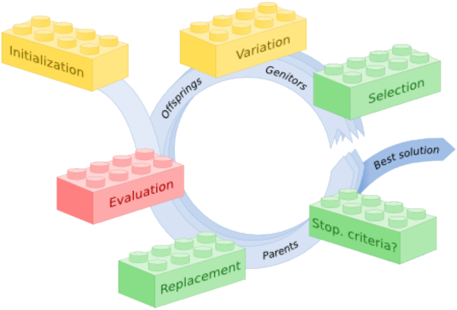
Minimize the loss function $\mathcal{L}(\theta_c, h_c)$.

Replacement

Elitism merge and deterministic tournament to keep 100 individuals.

Stop criteria

Reaching 400 generations.



Selection

Tournament between 2 individuals.

Learn ICN with Genetic Algorithms

Individual modeling

Vector of 29 bits:
one bit for each operations.

Initialization

Draw 100 individuals randomly.

Evaluation

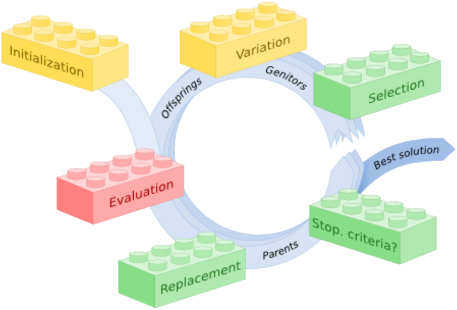
Minimize the loss function $\mathcal{L}(\theta_c, h_c)$.

Replacement

Elitism merge and deterministic tournament to keep 100 individuals.

Stop criteria

Reaching 400 generations.



Selection

Tournament between 2 individuals.

Variation

One-point crossovers.
One-flip mutations.

Outline

Tested constraints

5 major constraints:

- **All different**: variables must all be assigned to different values.
- **Ordered**: assignment of n variables $(x_1, \ldots, x_n)$ must be ordered, given a total order.
- **Linear sum**: equation $x_1 + x_2 + \ldots + x_n = p$ must hold.
- **No overlap**: variables represent tasks with a given length. A variable's value is its task starting time. No tasks must overlap.
- **Minimum**: the minimum value of an assignment must check a given numerical condition.

Three experimental protocols

Exp. 1: Scaling

Question: Do error functions learned over small spaces scale?

- Learn error functions over small spaces ($\simeq 500$ assignments),
- Test them over huge spaces ($\simeq 10^{200}$ assignments).

Three experimental protocols

Exp. 1: Scaling

Question: Do error functions learned over small spaces scale?

- Learn error functions over small spaces ($\simeq 500$ assignments),
- Test them over huge spaces ($\simeq 10^{200}$ assignments).

Exp. 2: Learning over incomplete spaces

Question: Can we learn efficient error function over incomplete spaces?

- Learning over 200 sampled assignments in large spaces ($\simeq 50.000$),
- Test them over huge spaces ($\simeq 10^{200}$ assignments).

Three experimental protocols

Exp. 1: Scaling

Question: Do error functions learned over small spaces scale?

- Learn error functions over small spaces ($\simeq 500$ assignments),
- Test them over huge spaces ($\simeq 10^{200}$ assignments).

Exp. 2: Learning over incomplete spaces

Question: Can we learn efficient error function over incomplete spaces?

- Learning over 200 sampled assignments in large spaces ($\simeq 50.000$),
- Test them over huge spaces ($\simeq 10^{200}$ assignments).

Exp. 3: Solving Sudoku with learned error functions

Question: Can a learned error function be used to solve an actual problem?

- Solve Sudoku with and without error functions.

Experimental result 1: Scaling

Constraints	median	mean	most freq.
all different	0	0.03	0 (97)
ordered	0.08	0.08	0.08 (100)
linear sum	0.01	0.05	0.01 (74)
no overlap	0.14	0.19	0.11 (50)
minimum	0	0.04	0 (88)

Table: Training error over small spaces (500 assignments).

all_diff	ord	lin_sum	no_ol	min
0	1.27	0.03	2.68	0

Table: Mean test error over 20,000 assignments in huge spaces.

Experimental result 1: Scaling

Constraints	median	mean	most freq.
all different	0	0.03	0 (97)
ordered	0.08	0.08	0.08 (100)
linear sum	0.01	0.05	0.01 (74)
no overlap	0.14	0.19	0.11 (50)
minimum	0	0.04	0 (88)

Table: Training error over small spaces (500 assignments).

all_diff	ord	lin_sum	no_ol	min
0	1.27	0.03	2.68	0

Table: Mean test error over 20,000 assignments in huge spaces.

Test spaces of size 10^{200}, but...

- Not easy to compute the Hamming error for Ordered and NoOverlap.
- Estimation of their Hamming error over spaces of size $\simeq 10^{15}$.

Experimental result 2: Incomplete spaces

Constraints	median	mean	most freq.
all different	0.44	0.44	0.44 (99)
ordered	0.44	0.46	0.44 (66)
linear sum	2.03	1.70	0.85 (37)
no overlap	2.33	2.39	2.29 (48)
minimum	0.59	0.59	0.59 (78)

Table: Training error over incomplete large spaces ($\simeq 50.000$ assignments).

all_diff	ord	lin_sum	no_ol	min
0	1.80	0.03	2.02	0

Table: Mean test error over 20,000 assignments in huge spaces.

Experimental result 3: Sudoku

Error Function	mean	median	std dev	min	max
no error functions	1044	764	727	250	3546
learned	383	331	268	57	1812
hard-coded	175	145	107	46	662
hand-crafted	149	125	107	26	608

Table: Run-times in milliseconds over 100 runs to solve Sudoku.

Rows in this table

1. No error functions (pure CN),
2. The most frequently learned error function for *All different* in Experiment 1 run through the ICN,
3. The same function but hard-coded in C++,
4. A hand-crafted error function (Petit et. al 2001).

Outline

Conclusion

- EFN power with CN model simplicity.
- Interpretable model.
- Scale!
- Can learn over incomplete spaces.
- No cherry-picked operations.
- Use it fully automatically or as a decision support system.

Conclusion

Conclusion

- EFN power with CN model simplicity.
- Interpretable model.
- Scale!
- Can learn over incomplete spaces.
- No cherry-picked operations.
- Use it fully automatically or as a decision support system.

Perspectives

- Need more diverse and expressive operations for very combinatorial constraints (Ordered, No overlap).
- Reinforcement learning to find error functions adapted to the solver.

Paper and source code

Get the paper!

arxiv.org/abs/2002.09811

Get the code!

github.com/richoux/LearningCostFunctions

Questions?

florian.richoux@polytechnique.edu

@FloRicx