Injecting Domain Knowledge in Neural Networks: a Controlled Experiment on a Constrained Problem

Mattia Silvestri[1] Michele Lombardi[1] and Michela Milano[1]

[1]University of Bologna
{mattia.silvestri4,michele.lombardi2,michela.milano}@unibo.it

1 Methods

In this section we formulate the methods that have been compared to inject knowledge provided by constraints propagation in the neural networks. Formally, given a constraint (or a collection of constraints) C, here we will treat its associated propagator as a multivariate function such that $C_j(x) = 1$ iff assignment $z_j = v_j$ has not been marked as infeasible by the propagator, while $C_j(x) = 0$ otherwise. The methods all share the following common structure:

$$L(x, y) = H\left(\frac{1}{Z} f(x), y\right) + \lambda L_{sbr}(x) \tag{1}$$

where H is the categorical cross-entropy, Z is the partition function, f is the output of the neural network (which employs the sigmoid activation function for each output neuron), x is the binary encoding of the PLS partial solution, y is the target assignment for the input partial solution and the scalar λ controls the balance between the crossentropy term H and the SBR term, i.e. the amount of trust we put in the incomplete domain knowledge. The methods differ for how L_{sbr} is formulated.

The first method discourages the neural network to make assignments which are provably infeasible according to the employed constraint propagator.

$$L_{sbr}(x) = \sum_{j=0}^{m-1} ((1 - C_j(x)) \cdot f_j(x)) \tag{2}$$

We will refer to this method as NEGATIVE in the following figures.
The others two methods both share the same idea which is the following: encouraging the neural network to give higher probability to local feasible assignments according to the constraints propagator. The first one uses the binary cross-entropy:

$$L_{sbr}(x) = \sum_{j=0}^{m-1} (C_j(x) \cdot log(f_j(x)) + (1 - C_j(x)) \cdot log(1 - f_j(x)) \tag{3}$$

and we will refer to it as BCE in the following plots. The second one employs the mean squared error:

$$L_{sbr}(x) = \sum_{j=0}^{m-1} (C_j(x) - f_j(x))^2 \tag{4}$$

and we will refer to it as MSE.

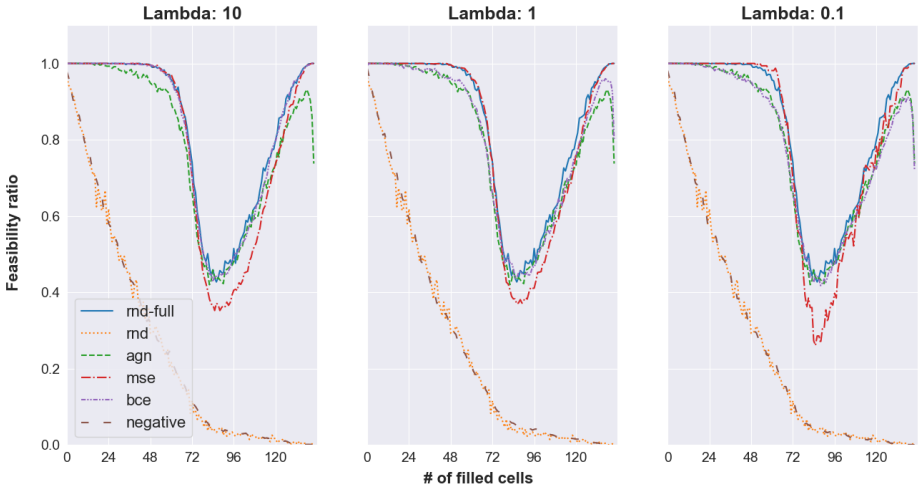

Fig. 1. Methods comparison for different λ values on the PLS-12 on a dataset generated from 10,000 solutions pool and with full constraints injection.

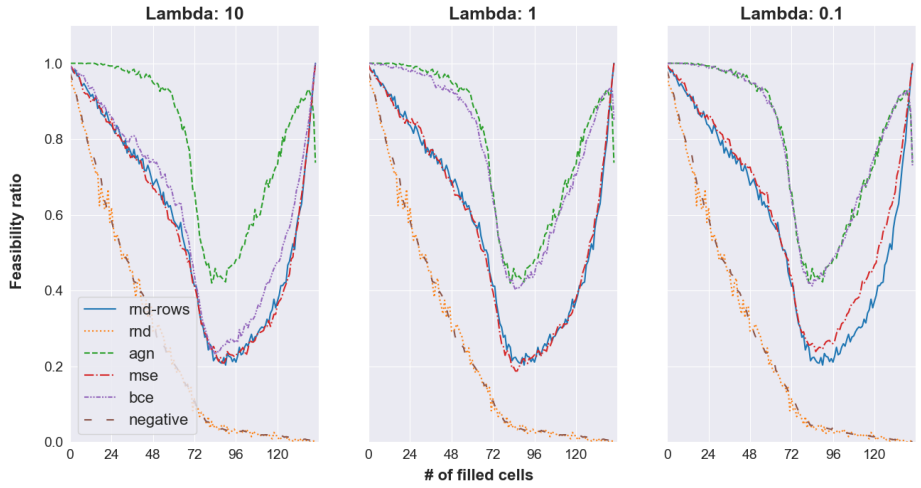

Fig. 2. Methods comparison for different λ values on the PLS-12 on a dataset generated from 10,000 solutions pool and with rows constraints injection.

2 Methods comparison and λ tuning

In this section we compare the different regularization approaches. The neural networks are trained on a set which is obtained via the random deconstruction procedure described in the paper, starting from a 10,000 solutions pool. We only focus on the greatest problem dimension (PLS-12). We also evaluate different values of λ to identify the best one for each method.

For each approach we train two different neural networks: a network trained with knowledge about row constraints and a network trained with knowledge about row and column constraints. For the first network, we employ the regularization method (and a Forward Checking propagator) to inject knowledge that both assigning a variable twice and assigning a value twice on the same row is forbidden, whereas for the second one we do the same, applying the Forward Checking propagator also to column constraints (i.e. no value can appear twice on the same column).
As also described in the paper, we compare them with a model-agnostic neural network (referred to AGN) and with methods that randomly choose an assignment with an uniform probability distribution but that can rely on a set of constraints C during the evaluation. Intuitively, this serves as an upper limit for the amount of knowledge that can be learned offline (since it comes from the same propagators). In particular, any improvement that may be observed over the behavior of the RND-FULL necessarily comes from information contained in the labels (i.e. the deconstruction of a solution), rather than from injecting knowledge about the propagator behavior.
We consider the two scenarios in which C is the set of the row constraints (RND-ROW) and the one in which C is the set of column and row constraints (RND-FULL).
We also compare the methods with a very pessimistic baseline, referred to RND, which randomly choose with uniform probability distribution among all the assignments (even the ones that are provably infeasible according to the constraints propagation).
We then produce the "feasibility plots". Since RND-FULL and RND-ROW are the only methods that can rely on **online** constraints propagation, we have highlighted them using solid lines.

For all the plots of the current and following sections, the regularization approaches can rely on the constraints propagation only **offline**, i.e. during training.
In fig. 1, we show results when all the constraints are employed by the forward checking constraints propagator, whereas in fig. 2 we do not propagate the columns constraints. The NEGATIVE approach does not work at all and its behaviour is hardly distinguishable by RND. We suspect that the reason is due to the fact that it encourages the network to keep the output as lowest as possible instead of discouraging the network to make provably infeasible assignments. The best λ values are 10 and 1 for respectively BCE and MSE. Low values of λ make the regularization not effective and they lead the methods to collapse to the model-agnostic neural network which does not perform well in scarcity of data or solutions, as we will see in the following sections. Our purpose is to distill the constraints propagator in the neural network's weights, finding a tradeoff between learning from correct (but sometimes scarce) knowledge and the incomplete one.

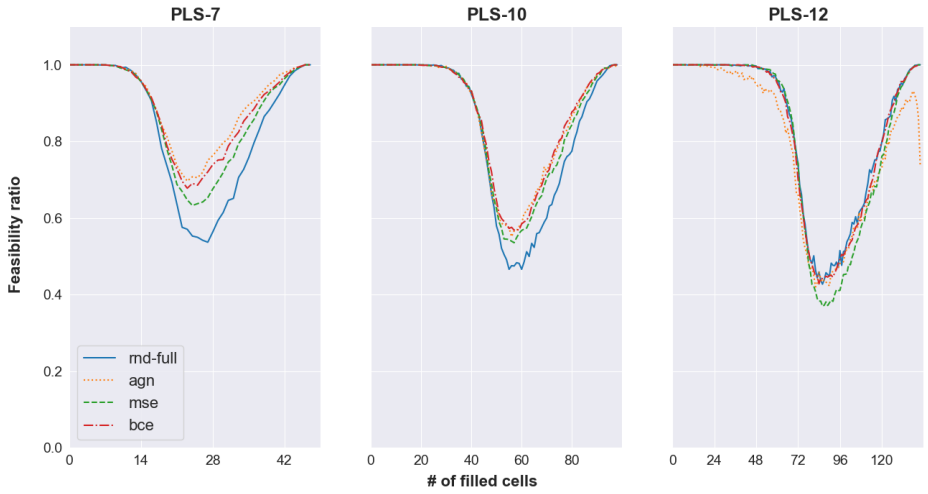

Fig. 3. Methods comparison for different problem dimensions and full constraints injection.

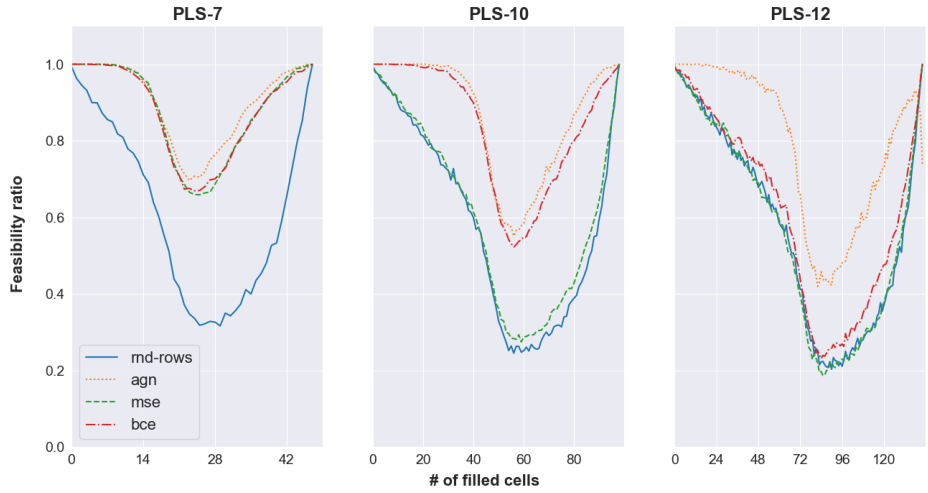

Fig. 4. Methods comparison for different problem dimensions and rows constraints injection.

3 Problem dimension

In this section we make an analysis comparing different problem dimension (PLS-7, PLS-10, PLS-12). Again, the training sets are generated starting from a pool of 10,000 solutions. The NEGATIVE approach is not further investigated due to its extremely bad results of the previous analysis. For BCE and MSE we only consider the best λ values (respectively 10 and 1). As we can see from fig. 3 and

fig. 4, if only a subset of constraints are known, the AGN approach still performs better. Relying on incomplete knowledge proves to be useful when the problem dimension becomes larger and all the constraints can be injected.

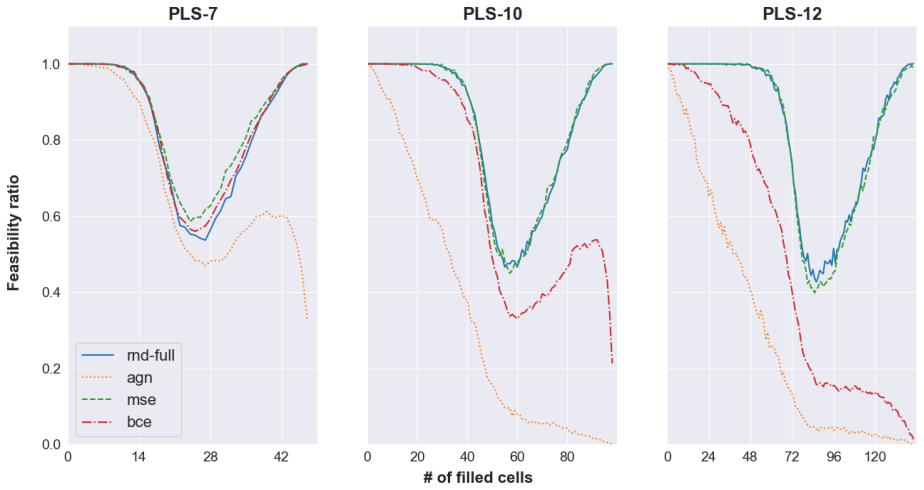

Fig. 5. Methods comparison for different problem dimensions and full constraints injection when the training set size is reduced to the 10% of the initial size.

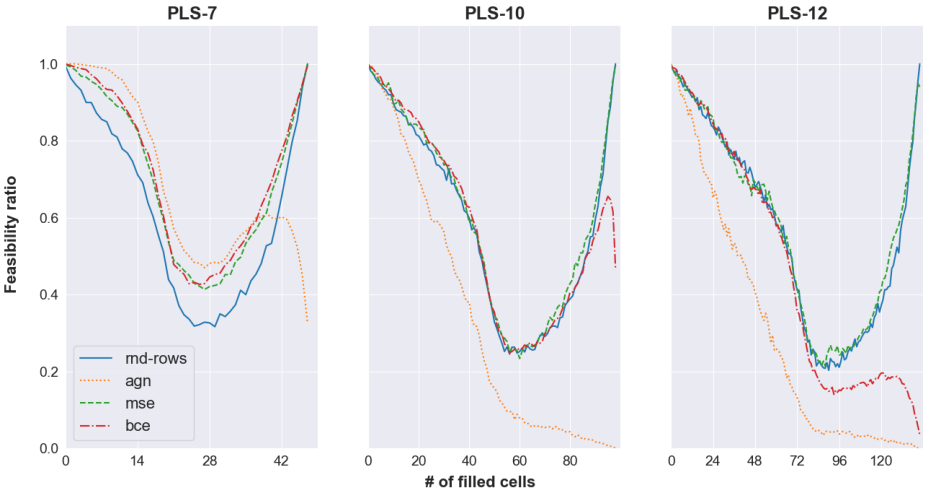

Fig. 6. Methods comparison for different problem dimensions and rows constraints injection when the training set size is reduced to the 10% of the initial size.

4 Training set size reduction

In this section we show results when the training set size is reduced to the 10% of its initial size for each problem dimension. As we can see in fig. 5 and fig. 6, the AGN performance drastically drop especially with a large problem dimension and the same can be stated for the BCE method. Instead, the MSE regularization

method is much more robust providing performance very similar to the random approach which can relies on the constraints propagation during test time.

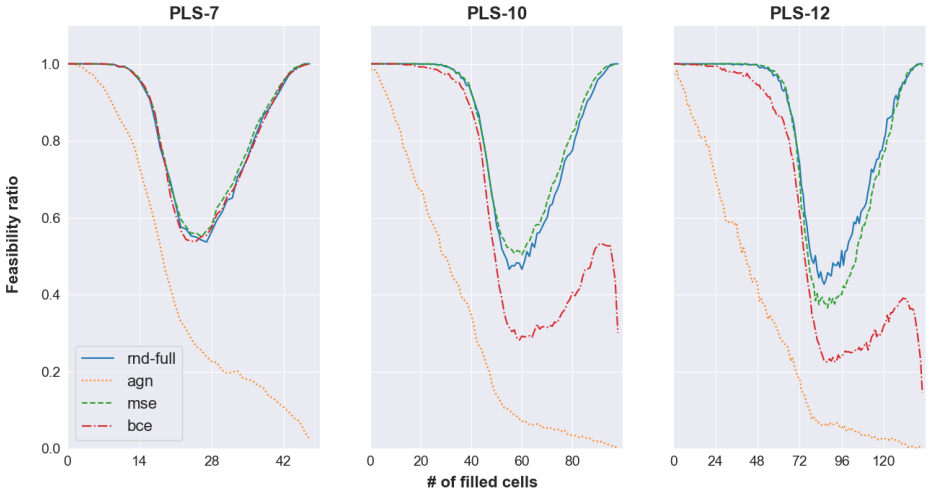

Fig. 7. Methods comparison for different problem dimension and full constraints injection when the solutions pool size is reduced from 10,000 to 100.

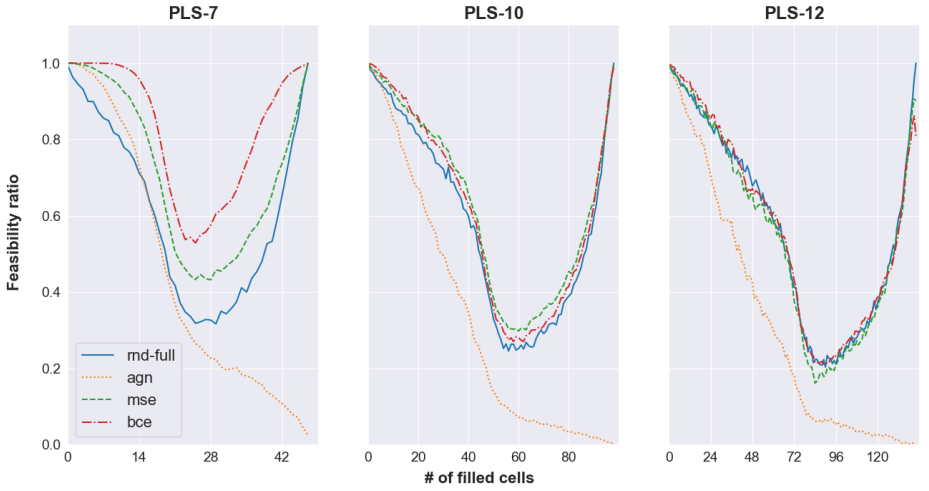

Fig. 8. Methods comparison for different problem dimension and rows constraints injection when the solutions pool size is reduced from 10,000 to 100.

5 Solutions pool reduction

As shown in fig. 7 and fig. 8, conclusions similar to the previous section ones can be drawn when the solutions pool size (from which the training set is generated) is decreased from 10,000 to 100. The MSE regularization approach is robust even when the problem dimension becomes larger and the empirical knowledge is scarce. Results demonstrate how the method is effectively distilling the con-

straints propagators in the network's weights so that that can be employed even when it is not feasible at test time.