Appendix A: Proof of Lemma 1

The proof proceeds in two parts. First, we show that if the combination of active neurons is known, the network can be equivalently represented by a modified architecture where activation functions are replaced by identity functions, and the corresponding weights are adjusted accordingly. In the second part, we demonstrate that, under identity activation functions, the output of the network becomes a linear transformation of the input features. Specifically, we show that the output can be expressed as a linear combination of the input in the form: $\bar{\mathbf{w}}^{\omega}\boldsymbol{\mathcal{X}} + \bar{b}^{\omega}$, where

$$\bar{\mathbf{w}}^{\omega} = \prod_{k=0}^{K} \mathbf{W}^{\omega,k} \qquad \bar{b}^{\omega} = \mathbf{b}^{\mathbf{K}} + \sum_{k=0}^{K} \left(\prod_{l=k+1}^{K} \mathbf{W}^{\omega,l} \right) \mathbf{b}^{k}. \tag{21}$$

Part 1: Assume the activation pattern ω is known. For each layer $k \in \{0, \dots, K\}$, we define the modified weight matrix $\mathbf{W}^{\omega,k}$ by:

$$W_{ji}^{\omega,k} = \begin{cases} w_{ji}^{k} & \text{if } \mathbf{w}_{j,:}^{k}\mathbf{a}^{k} + b_{j}^{k} > 0, \\ 0 & \text{otherwise,} \end{cases}$$

This construction is equivalent to applying the ReLU activation after each layer since the ReLU function sets to zero the negative values. By setting to 0 the corresponding weights for inactive neurons, we propagate through the network as if ReLU were replaced by the identity function, under the fixed activation pattern.

Part 2: Assume the activation function is the identity. We show by induction that the value of each neuron in layer $k \in \{1, \dots, K+1\}$ is a linear transformation of the input: $\mathbf{a}^{k} = \bar{\mathbf{w}}^{\omega,k-1}\boldsymbol{\mathcal{X}} + \bar{\mathbf{b}}^{\omega,k-1}$ where:

$$\bar{\mathbf{w}}^{\omega,k} = \prod_{m=0}^{k} \mathbf{W}^{\omega,m}, \quad \bar{\mathbf{b}}^{\omega,k} = \mathbf{b}^{k} + \sum_{m=1}^{k} \left(\prod_{l=m}^{k} \mathbf{W}^{\omega,l} \right) \mathbf{b}^{m-1}.$$

The values of the neurons at the first hidden layer are $\mathbf{a}^{1} = \bar{\mathbf{w}}^{\omega,0}\boldsymbol{\mathcal{X}} + \bar{\mathbf{b}}^{\omega,0}$, which matches the claimed form with:

$$\bar{\mathbf{w}}^{\omega,0} = \mathbf{W}^{\omega,0}, \quad \bar{\mathbf{b}}^{\omega,0} = \mathbf{b}^{0}.$$

Assume the result holds for layer k, then the values of the neurons in layer $k+1$ are:

$$\begin{aligned} \mathbf{a}^{k+1} &= \mathbf{W}^{\omega,k}\mathbf{a}^{k} + \mathbf{b}^{k} \\ &= \mathbf{W}^{\omega,k} \left(\prod_{m=0}^{k-1} \mathbf{W}^{\omega,m}\boldsymbol{\mathcal{X}} + \mathbf{b}^{k-1} + \sum_{m=1}^{k-1} \left(\prod_{l=m}^{k-1} \mathbf{W}^{\omega,l} \right) \mathbf{b}^{m-1} \right) + \mathbf{b}^{k} \\ &= \left(\prod_{m=0}^{k} \mathbf{W}^{\omega,m} \right) \boldsymbol{\mathcal{X}} + \mathbf{b}^{k} + \sum_{m=1}^{k} \left(\prod_{l=m}^{k} \mathbf{W}^{\omega,l} \right) \mathbf{b}^{m-1}. \end{aligned}$$

Thus, the result also holds for layer $k+1$, completing the inductive step.

Appendix B: Proof of Lemma 1 for ICNN

The proof of Lemma 1 can be extended to ICNNs by including weights $\hat{\mathbf{w}}^{m}$. Assume the activation function is the identity. We show by induction that the value of each neuron in layer $k \in \{1, \dots, K+1\}$ is a linear transformation of

the input: $\mathbf{a}^k = \bar{\mathbf{w}}^{\omega,k-1}\boldsymbol{\mathcal{X}} + \bar{\mathbf{b}}^{\omega,k-1}$ where:

$$\bar{\mathbf{w}}^{\omega,k} = \hat{\mathbf{w}}^k + \sum_{m=1}^{k}\left(\prod_{l=m}^{k}\mathbf{W}^{\omega,l}\right)\hat{\mathbf{w}}^{m-1}, \quad \bar{\mathbf{b}}^{\omega,k} = \mathbf{b}^k + \sum_{m=1}^{k}\left(\prod_{l=m}^{k}\mathbf{W}^{\omega,l}\right)\mathbf{b}^{m-1}.$$

The values of the neurons at the first hidden layer are $\mathbf{a}^1 = \bar{\mathbf{w}}^{\omega,0}\boldsymbol{\mathcal{X}} + \bar{\mathbf{b}}^{\omega,0}$, which matches the claimed form with:

$$\bar{\mathbf{w}}^{\omega,0} = \hat{\mathbf{w}}^0, \quad \bar{\mathbf{b}}^{\omega,0} = \mathbf{b}^0.$$

Assume the result holds for layer k, then the values of the neurons at layer $k+1$ are:

$$\begin{aligned}
\mathbf{a}^{k+1} &= \mathbf{W}^{\omega,k}\mathbf{a}^k + \mathbf{b}^k + \hat{\mathbf{w}}^k\boldsymbol{\mathcal{X}} \\
&= \mathbf{W}^{\omega,k}\left[\left(\hat{\mathbf{w}}^{k-1} + \sum_{m=1}^{k-1}\left(\prod_{l=m}^{k-1}\mathbf{W}^{\omega,l}\right)\hat{\mathbf{w}}^{m-1}\right)\boldsymbol{\mathcal{X}} + \mathbf{b}^{k-1} + \sum_{m=1}^{k-1}\left(\prod_{l=m}^{k-1}\mathbf{W}^{\omega,l}\right)\mathbf{b}^{m-1}\right] + \mathbf{b}^k + \hat{\mathbf{w}}^k\boldsymbol{\mathcal{X}} \\
&= \left[\left(\mathbf{W}^{\omega,k}\hat{\mathbf{w}}^{k-1} + \sum_{m=1}^{k-1}\left(\prod_{l=m}^{k}\mathbf{W}^{\omega,l}\right)\hat{\mathbf{w}}^{m-1} + \hat{\mathbf{w}}^k\right)\boldsymbol{\mathcal{X}} + \mathbf{b}^k + \mathbf{W}^{\omega,k}\mathbf{b}^{k-1} + \sum_{m=1}^{k-1}\left(\prod_{l=m}^{k}\mathbf{W}^{\omega,l}\right)\mathbf{b}^{m-1}\right] \\
&= \left(\hat{\mathbf{w}}^k + \sum_{m=1}^{k}\left(\prod_{l=m}^{k}\mathbf{W}^{\omega,l}\right)\hat{\mathbf{w}}^{m-1}\right)\boldsymbol{\mathcal{X}} + \mathbf{b}^k + \sum_{m=1}^{k}\left(\prod_{l=m}^{k}\mathbf{W}^{\omega,l}\right)\mathbf{b}^{m-1}.
\end{aligned}$$

Thus, the result also holds for layer $k+1$, completing the inductive step.

Appendix C: Number of combinations for ICNN

In this proof, we are interested in enumerating all combinations of active neurons in an input convex neural network. In a convex ReLU network, an active neuron outputs a positive value, and we can find a path that connects this neuron to at least one input feature while passing through active neurons only. In the following, a layer is said to be active if at least one of its neurons is active.

First, it is evident that, excluding the unique scenario where all neurons are inactive, an active layer k from a neural network with ReLU activation function presents $2^{N_k} - 1$ distinct combinations. In a convex ReLU network with more than one hidden layer, each layer $k \in \{2, \dots, K\}$ is active if and only if layer $k-1$ is active. Hence, the number of combinations of a ReLU network is equal to:

$$|\Omega| = 1 + \prod_{k=1}^{K}(2^{N_k} - 1)$$

However, in input convex neural networks, the above assumption does not hold since a neuron can become active after receiving a positive weighted sum from the input layer.

There exist other combinations in which the first hidden layers are active, but at least one of the following layers is inactive. These combinations can be ignored because they are equivalent to the combination where the first layers are inactive. This can be illustrated by the neural network of Figure 2e, but considering neurons a_1^1 or a_2^1 as active neurons. In this case, the prediction of the resulting ICNN is equal to the same networks with no active neurons in the first hidden layers. Therefore, we only consider combinations where an active layer results in the following layers being active. In addition to the combinations of a multilayer ReLU network, we need to consider all the combinations

resulting from a succession of active layers $k \in \{l, \dots, K\}, l > 1$, for which layer $k-1$ is not active. Thus, the number of possible combinations of an input convex ReLU network is equal to:

$$|\Omega| \; = \; 1 + \sum_{l=1}^{K} \prod_{k=l}^{K} (2^{N_k} - 1).$$

Note that, depending on the weights and biases in the network, this formula represents an upper bound on the actual number of combinations of an input convex neural network. For instance, a neuron might remain perpetually active if it receives strictly positive biases and weights, and all the combinations involving this neuron as inactive can thus be deleted from Ω.

For convex Maxout networks, the number of possible combinations is given by enumerating all the combinations of feature maps for each Maxout unit. Given a non-negative (or an input convex) Maxout network with K hidden layers, each Maxout unit belonging to layer $k \in K$ returns the highest value among F_k feature maps, which gives $(F_k)^{N_k}$ combinations of active neurons in each layer, and $\prod_{k=1}^{K} (F_k)^{N_k}$ for the whole network.

Appendix D: Lot-sizing instance generation

The lot-sizing instances are generated as described in Wolosewicz et al. (2015). For each item, the production, holding, and backlogging costs are identical between the instances, and we consider respective values of 4 for the production costs, 1 for the holding costs, and 5 for the backlogging costs. In this procedure, the available capacities are computed based on the demand, processing times, and setup times of each product and multiplied by a reduction factor:

$$C_t = cap \sum_{i \in \mathcal{I}} \sum_{o=1}^{O_i} \sum_{k \in M_{io}} \frac{1}{|M_{io}|} p^u_{iok} d_{it} + s^{mean}_k \quad \forall t \in \{1, \dots, T\},$$

where O_i is the number of operations associated with item $i \in \mathcal{I}$, M_{io} is the subset of machine that can perform operation $o \in \{1, \dots, O_i\}$ of item i, p^u_{iok} represents the processing time of operation o of item i when performed on machine $k \in M_{io}$, s^{mean}_k is the average setup times occurring on machine k and cap is the reduction factor. In our experiments, we observed that the reduction factor used in Wolosewicz et al. (2015) led to lot-sizing instances with very large capacities, and we used a factor of 0.35 for scheduling sizes 10×10 and 20×5 instead of 0.55 as in Wolosewicz et al. (2015).

Appendix E: Prediction performance of different Maxout architectures

Table 7 shows the performance of three convex Maxout networks with different architectures. The first network represents a single layer of one Maxout unit with 15625 feature maps, which can be decomposed into 15625 combinations. The second network can also be modeled with 15625 combinations but consists of a single layer of 2 Maxout units, each returning the maximum across 125 feature maps. The third network is similar to the second but includes 50 feature maps instead of 125, for a total of 2500 combinations. Note that `CMN` from Table 1 with architecture (10,10,10) can also be modeled with 15625 combinations. The results of Table 7 show no significant impact of the architecture of Maxout networks on the prediction. In addition, Maxout networks with a large number of Maxout units require fewer evaluations of the maximum values across feature maps, which can reduce the computational time of the relaxation approach proposed in this work. Although integrating small neural network architectures into an MILP significantly reduces the computational burden, the performance of small networks may deteriorate when trained with iterative adversarial sampling. The number of combinations in a convex neural network is a good indicator of the learning capabilities.

Table 7 Prediction performance of different architectures of Maxout networks

Size	Metric	CMN 15625 : 1	CMN 125 : 2	CMN 50 : 2
6×6	Accuracy (%)	96.6	97.5	96.7
	Precision (%)	100.0	100.0	100.0
	Recall(%)	93.2	95.0	93.4
	MAPE(%)	2.3	1.9	2.2
	NBUnder(%)	0.204	0.184	0.172
	MaxUnder(%)	1.4	1.5	1.3
10×10	Accuracy (%)	94.1	94.6	93.7
	Precision (%)	99.9	100.0	100.0
	Recall(%)	88.2	89.2	87.5
	MAPE(%)	3.4	3.1	3.7
	NBUnder(%)	1.324	0.78	0.468
	MaxUnder(%)	3.8	2.4	3.0
20×5	Accuracy (%)	90.1	89.1	88.0
	Precision (%)	99.9	99.9	100.0
	Recall(%)	80.4	78.2	76.1
	MAPE(%)	5.5	6.1	6.9
	NBUnder(%)	2.62	1.396	1.22
	MaxUnder(%)	20.7	13.1	6.2

Appendix F: Conjunctive graph formulation

Wolosewicz et al. (2015) proposed a formulation for integrated lot-sizing and fixed scheduling problems based on the conjunctive graph of the scheduling. In job shop scheduling, the scheduling problem can be represented as a disjunctive graph, where each conjunctive arc indicates the precedence between operations from the same job, and the disjunctive arcs represent the precedence between operations performed on the same machine. By fixing the scheduling sequence, the corresponding graph becomes a conjunctive graphs and the makespan thus is equal to the length of the longest paths in the graph, where each path is a sequence of operation (i,o,k) with $i \in \mathcal{I}, o \in \{1,\dots,O_i\}$ and $k \in M_{io}$. Therefore, by considering C the set of path in the conjunctive graph, function g can be defined by the MILP formulation as follows (Wolosewicz et al. 2015):

$$\sum_{(i,o,k)\in c} p^u_{iok} X_{it} + s^k_i Y_{it} \leq C_t, \qquad t = 1,\dots,T, \quad c \in C. \tag{22}$$

Constraints (22) ensure that all paths in the conjunctive graph respect the capacity in period t. Note that, as the sequence is fixed, setup times s^k_i are no longer sequence-dependent, and depend only on item i. In this formulation, constraints (15) can be relaxed using the same approach as the one proposed in this work. Hence, Wolosewicz et al. (2015) proposed a Lagrangian heuristic that repairs the solution by shifting production from source periods to destination periods. However, we observed that this procedure struggles to find feasible production plans when capacity is tight, so we chose a repair procedure based on the problem's mathematical formulation.