A GNN-aided fix-and-resolve approach for the Lot Sizing Problem

Mathieu Lerouge, Andrea Lodi, Enrico Malaguti, Michele Monaci, Filippo Focacci

A GNN-aided fix-and-resolve approach for the Lot Sizing Problem

Mathieu Lerouge[1,2], Andrea Lodi[1], Enrico Malaguti[1], Michele Monaci[1], Filippo Focacci[2]

[1] Università di Bologna, Bologna, Italy
{mathieu.lerouge, andrea.lodi, enrico.malaguti, michele.monaci}@unibo.it
[2] DecisionBrain, Paris, France
{mathieu.lerouge, filippo.focacci}@decisionbrain.com

Keywords : *Reoptimization, Machine Learning, Graph Neural Network, Lot Sizing Problem.*

1 Context

In industrial production planning, the Lot Sizing Problem (LSP) aims at determining a production plan that minimizes operational costs while satisfying demand and capacity constraints, over a planning horizon discretized into periods. While optimal or near-optimal solutions are typically computed solving a Mixed-Integer Linear Programming (MILP) model over hours, unexpected perturbations — such as machine breakdowns or sudden demand peaks — may make these solutions infeasible, shortly before or during execution.

In such situations, decision-makers face a dual challenge: they must find a new feasible good-quality solution within a short time (*e.g.* seconds or minutes) to quickly react to the perturbation, while ensuring stability by avoiding important deviations from the original solution. Resolving the full MILP model would be computationally prohibitive. A simple repair heuristic would lead to a poor-quality solution. Consequently, there is a need for methods that can leverage the knowledge of the original solution to efficiently explore its neighborhood and find a feasible good-quality revised solution.

2 Related work on reoptimization and ML

Such methods fall within the domain of reoptimization, where a new solution is computed for a modified instance based on an existing optimal (or near-optimal) solution.

The recent literature has successfully integrated Machine Learning (ML) into MILP-based reoptimization approaches. Xavier *et al.* [6] use ML to initialize a separation-like method with an ML-predicted relevant group of constraints to consider at the beginning of the solving process. However, the applicability of this method is limited to problems solvable through separation techniques. Lodi *et al.* [4] and Morabit *et al.* [5] employ ML to reduce the feasible solution space of the reoptimization problem. Specifically, [4] predicts bounds on sums of binary variables, while [5] predicts whether specific binary variables should be fixed. However, a limitation of these approaches is their reliance on fixed-dimension inputs, requiring different ML models for instances of different sizes. This restricts their applicability in varying production environments (*e.g.* varying number of items) among others. Our work overcomes this limitation by leveraging Graph Neural Networks (GNNs) [2], which generalize to variable-sized inputs. It follows up on the work "ML-guided reoptimization applied to LSP" presented last year.

3 A GNN-aided fix-and-resolve approach

We introduce a GNN-aided "fix-and-resolve" method applied to the LSP under machine breakdown perturbations. Our method works essentially by identifying which setup binary decision variables in the MILP formulation should be maintained free and which ones should be fixed to their values in a repaired solution $\mathcal{S}^r$ (quickly generated from the original solution by canceling the productions on periods where the machine is no longer available).

First, to ensure operational stability and limit the search space, we first consider a neighborhood constraint enforcing that only κ setup variables related to the first τ periods can have values different from their values in $\mathcal{S}^r$ (where κ and τ are fixed parameters). Then, to identify which setup variables within

the first $\mathcal{T}$ periods are most likely to change, we design a GNN model (see Fig. 1). We represent the LSP instance, the original solution, and the perturbation as a graph consisting of: three types of nodes (Machine-Period, Item-Period and Production nodes) associated to vectors of features (*e.g.* machine capacity before/after perturbation, machine capacity utilization or item demand); various types of edges connecting the nodes together (representing resource usage, temporal flows, and competition for capacity).

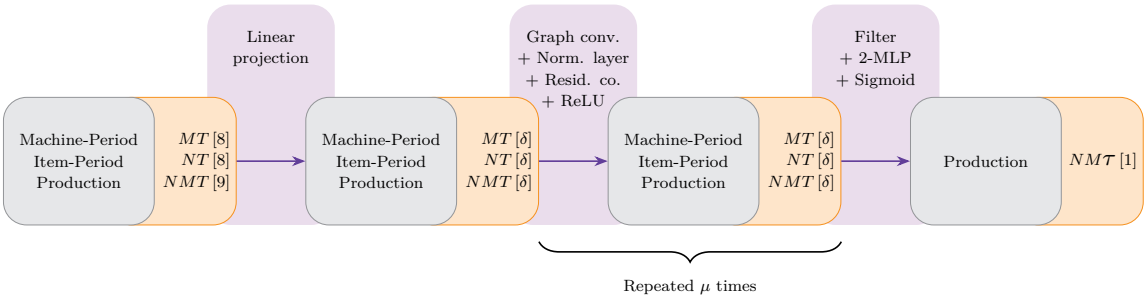

FIG. 1: Schematic view of the GNN architecture. The GNN processes the input feature graph through first a linear projection in an embedding dimension δ and then μ graph convolution blocks to finally output a change-likeliness score σ_{ijt} for each Production node.

The GNN processes this graph through a stack of graph convolution blocks. Each block learns distinct message-passing functions for different edge types, capturing the complex dependencies between machines, items and periods. Residual connections and normalization layers are employed to stabilize training. The network outputs a score $\sigma_{ijt} \in [0, 1]$ for each Production node, *i.e.* for each setup variable – since Production nodes coincide with setup variables. This score σ_{ijt} represents the likelihood that the production status of item i on machine j at period t should differ from the repaired solution $\mathcal{S}^r$. Given these scores, following a rank-based selection strategy, we identify the λ setup variables with the highest predicted scores (where λ is a multiple of κ). In the MILP formulation, we maintain the λ setup variables free and fix the rest.

The model is trained in a supervised manner using labels derived from near-optimal reconfigured solutions obtained via extensive solver runs. Given the strong imbalance between the coefficients 0 and 1 in the score vector — since only about 1% of setup variables typically change value — we train the GNN using a focal loss function [3] rather than a standard binary cross-entropy. This forces the GNN to focus on the hard-to-classify minority of coefficients 1.

Numerical results show that our approach generalizes across different instance sizes. Our dataset, inspired by the benchmark used in [1], consists of instances with 2 to 4 machines, 30 to 40 items and 30 periods. Considering a 10 work-unit[1] time limit, our method achieves an average cost improvement of 23% over $\mathcal{S}^r$, compared to 13% for a standard resolve baseline — with $\mathcal{S}^r$ used as warm-start in both cases.

References

[1] M. Charles. Modeling and solving complex multi-item lot-sizing problems with inventory constraints. *PhD thesis*, Université de Lyon, 2021.

[2] M. Gasse, D. Chételat, N. Ferroni, L. Charlin, and A. Lodi. Exact Combinatorial Optimization with Graph Convolutional Neural Networks. *NeurIPS*, 32, 2019.

[3] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár. Focal loss for dense object detection. In *ICCV*, pages 2980–2988, 2017.

[4] A. Lodi, L. Mossina, and E. Rachelson. Learning to handle parameter perturbations in Combinatorial Optimization: An application to facility location. *EURO J. Transp. Logist.*, 9(4):100023, 2020.

[5] M. Morabit, G. Desaulniers, and A. Lodi. Learning to repeatedly solve Routing Problems. *Networks*, 83(3):503–526, 2024.

[6] Á. S. Xavier, F. Qiu, and S. Ahmed. Learning to Solve Large-Scale Security-Constrained Unit Commitment Problems. *INFORMS Journal on Computing*, 33(2):739–756, 2021.

[1] According to Gurobi's documentation, work units approximate the solver's effort, roughly corresponding to one second of single-threaded execution, though their exact value depends on hardware and model characteristics. Unlike wall-clock time, work units isolate the solver's performance from system-level variability, such as background processes.