Available online at www.sciencedirect.com

ScienceDirect

IFAC PapersOnline CONFERENCE PAPER ARCHIVE

IFAC PapersOnLine 59-10 (2025) 482–487

Towards Explainable Optimization of Production System Configurations ⋆

András Kovács * **Zsombor Szádoczki** *,** **Dávid Karnok** *

* *HUN-REN Institute for Computer Science and Control (SZTAKI), Kende u. 13-17, H-1111 Budapest, Hungary*
** *Department of Operations Research and Actuarial Sciences, Corvinus University of Budapest, 1093 Fővám tér 8., Budapest, Hungary*
{*andras.kovacs,zsombor.szadoczki,david.karnok*}*@sztaki.hu*

Abstract: The critical decisions related to production system configuration are often supported by mathematical optimization tools, such as mixed-integer linear programming (MILP) models built into automated system design software. However, users may have concerns about the computed optimal solution–or about the absence of a solution–arising from discrepancies between the mathematical model and their understanding of the problem, or from the flexibility of input parameters. Thus, to support the decision-making process, it is crucial to make optimization explainable, i.e., to reinforce the understanding of the user why a given solution is recommended. This paper proposes an interactive procedure to solve this problem, where the decision-maker asks consecutive "why not c" type questions, where c is a feature of the solution encoded in a set of constraints on the decision variables of the MILP. Special focus is given to the case where infeasibility must be explained. Two alternative search procedures were implemented to find all possible explanations of infeasibility. The proposed approach was validated on a medium-scale sample instance of the system configuration problem with 17 tasks requiring 13 resources for their execution. The gained explanations can help identify the parameters and constraints that require special attention by the user.

Keywords: Manufacturing modeling, assembly line balancing, mathematical programming, optimization, explainable optimization.

1. INTRODUCTION

1.1 Motivation

Designing complex production systems requires making decisions based on the complex understanding of a number of constraints and objectives related to different fields of engineering. Models based on artificial intelligence (AI) or classical mathematical methods are frequently applied to support this process. However, it often happens that the mathematical model does not completely match the end users' understanding of the problem. Moreover, the algorithm for computing the solution may not be transparent for the end user with a background in engineering, rather than mathematical optimization. Finally, in early-stage production system design, parameters are uncertain, or even can be controlled (e.g., task durations are estimated according to some methodology, but never tested in physical reality; yet, if needed, they can be tuned by different methods, including equipment selection, geometrical layout, process parameters, etc.). Thus, to support the decisions of the end users, as well as to build their trust in automated problem solving, the level of explainability, that is the "level of understanding how the AI-based system ... came up with a given result" (ISO/IEC, 2020), is one of the most crucial aspects.

Explainable Artificial Intelligence (XAI) focuses on making the predictions and recommendations of automated systems clear and understandable to human experts. While most effort has been invested into explaining the results of black-box machine learning approaches (Linardatos et al., 2021; Mersha et al., 2024), explanations are equivalently important for classical mathematical optimization techniques. Even though these approaches are much better interpretable for the expert who designed them, the end user with a different background may still perceive them as black boxes and need support in understanding why a given solution is recommended.

1.2 Literature Review

The literature of explainability in the field of mathematical optimization is rather scarce, however, significant advancements have been made recently. (Goerigk and Hartisch, 2023) introduced a generic optimization framework to construct interpretable decision rules for optimization prob-

⋆ This research was supported by the TKP2021-NKTA-01 and the 2020-1.2.3-EUREKA-2022-00014 grants.

10.1016/j.ifacol.2025.09.083

lems in the form of decision trees, sacrificing exact optimality for interpretability. (Ott and Jäkel, 2024) proposed techniques for explaining the optimal solutions of linear programs to non-expert users. Contrastive explanations for combinatorial optimization problems, comparing the optimal solution to given alternatives, are investigated in (Erwig and Kumar, 2024), with illustrations on various graph problems. In (Aigner et al., 2024), a data-driven approach is proposed for computing explainable, close-to-optimal solutions for optimization problems that resemble favorable past solutions but differ from unfavorable previous solutions. The explainability of AI planning was investigated in (Fox et al., 2017). In the same context, (Krarup et al., 2021) proposes contrastive explanations for answering "why A rather than B" type questions. The approach is extended to explaining the differences in the quality of the two plans in (Krarup et al., 2024). (Chakraborti et al., 2017) argue that explanations must focus on the differences between the planning model and the user's understanding of the problem, and interpret explanations as a *model reconciliation problem* (MRP), looking for minimal modifications of the user model in such a way that the optimal solution of the planning model becomes optimal for the user as well. (Nguyen et al., 2020) propose an answer set programming approach for solving the MRP.

In this paper, the focus is on the production system configuration problem for flow systems. This problem can be seen as a major extension of the *assembly line balancing problem* (ALBP) (Becker and Scholl, 2006) with detailed resource requirements. For a recent survey of ALBP, see (Boysen et al., 2022). In (Battaïa et al., 2020), a mathematical programming approach is proposed for a configuration problem with a very detailed resource model, tailored to flow lines with reconfigurable machine tools. (Tsutsumi et al., 2022) deals with the integration of production system configuration and task sequencing using a mixed-integer linear programming (MILP) model. (Dobrovoczki et al., 2024) proposes a Benders decomposition method for solving a generic configuration-and-layout problem.

1.3 Contributions

The insight provided by explanations on why given solutions are recommended by automated optimization software is particularly valuable and relevant in early-stage production system configuration where a considerable share of the input parameters are uncertain or negotiable. Despite this, to the best of the authors' knowledge, explainability has not been studied in the field of production system configuration.

Accordingly, the main contribution of this paper is proposing an interactive explainability framework for production system configuration, presenting how the baseline optimization model must be extended to compute the desired explanations, as well as illustrating the proposed approach on a medium-scale sample problem.

1.4 Structure of the Paper

This paper is structured as follows. After a formal problem definition (Section 2), a MILP model of the production system configuration problem is formulated in Section 3. Section 4 introduces the proposed approach to explaining the optimal solution. The approach is demonstrated via an illustrative example in Section 5. Finally, conclusions are drawn and directions for future research are proposed in Section 6.

2. PROBLEM DEFINITION

The investigated production system configuration problem addresses the design of a single serial *production line* to manufacture multiple products according to the given process plans. The process plan of each product consists of a fully ordered sequence of *tasks*. The rough structure of the line is specified by the customer and also given in the input: the line consists of multiple *stations* along a conveyor, and tasks must be executed at predefined *locations* within a given station. This defines a three-level hierarchy of locations, stations, and the single line. The optimization problem consists in selecting the specific resources to install at the different nodes of this hierarchy most efficiently.

Each task *requires* one or multiple *functions* for its execution; e.g., a screw driving task requires the screw driving function. Accordingly, a resource that *provides* the given function must be selected from a predefined *resource library*, installed into the line, and assigned to the task; e.g., the screw driving function requirement can be satisfied by an electric or a pneumatic screwdriver. In a similar fashion, the selected resources themselves can require further functions, e.g., the pneumatic screwdriver requires the compressed air function, which can be provided by one of the available air compressors. Each function requirement, either stemming from a task or a resource, must be satisfied by a resource installed at the source node of the requirement, or a node above the origin of the requirement in the hierarchy. For instance, to satisfy the compressed air function requirement stemming from the pneumatic screwdriver installed at the location of the screw driving task, the air compressor must be installed at that specific location, at the given station, or on the line level. In this case, it might be worth installing the compressor on the line level, since then it can provide compressed air to all other resources in the entire line. For each resource in the library, it is defined whether it can be installed on the location, on the station, or on the line levels of the hierarchy. The investment cost of each resource is also given in the input.

The input also contains a prescribed cycle time for each product. The cycle time of the entire line is determined by the slowest station, whereas the cycle time of a station is defined by the sum of the durations of the tasks executed sequentially on the product within the station. Task durations depend on the assigned resources. Moreover, synchronization constraints between pairs of stations are also imposed, to capture the fact that multiple parts are passed between the stations at specific points in time within a single cycle. Each synchronization constraint states that the total duration of two subsets of tasks must be equal.

Then, the production system configuration problem involves determining the resources installed at the different nodes of the resource hierarchy (binary variable x_{isl} in-

Table 1. Notations.

Indices	
j	Product index
t	Task index
i	Resource index
s	Station index
l	Location index
f	Function index
k	Synchronization index
$j(t)$	Product belonging to task t
$s(t)$	Station belonging to task t
$l(t)$	Location of task t
Input parameters	
Λ	Set of resources applicable on line level
Σ	Set of resources applicable on station level
Π	Set of resources applicable on location level
R_t	Functions required by task t
Q_i	Functions required by resource i
P_i	Functions provided by resource i
d_{ti}	Duration of task t when executed by resource i
T_j	Cycle time limit of product j
$\left(\tau_1^k, \tau_2^k\right)$	For synchronization k, task sets τ_1^k and τ_2^k must be synchronized
c_i	Investment cost of resource i
Decision variables and objective	
δ_t	Actual duration of task t in the configuration
ρ_{fsl}	Function f is required at location l of station s
x_{isl}	Resource i is installed at location l of station s
y_{ti}	Task t is assigned to resource i
C	Investment cost

dicates whether an instance of resource i is installed at station s, location l), as well as the resource assigned to each task (binary variable y_{ti} indicates if task t is executed by resource i) in such a way that all the above constraints are satisfied and the total investment cost is minimized. The notation is summarized in Table 1. In the representation of the three-level resource hierarchy of line, stations and locations, station index $s > 0$ and location index $l = 0$ indicate the node corresponding to station s, whereas $s = 0$ and $l = 0$ stand for the line.

It is emphasized that the above description presents the baseline problem, which can be easily formulated and solved as a MILP, as detailed in Section 3. The challenge addressed in this paper is explaining the computed solution to the user of the automated system design software.

3. MATHEMATICAL MODEL FOR SYSTEM CONFIGURATION

The mixed-integer linear programming (MILP) model of the system configuration problem is presented in Figure 1. The objective is to minimize the investment cost (1). Constraint (2) ensures that every task is assigned to exactly one resource. Constraints (3), (4) and (5) state that each resource can be installed only on the appropriate levels of the resource hierarchy, i.e., on the line, on the station, and on the location levels, respectively. Line (6) encodes that no location exists outside the stations. Constraint (7) states that a resource can be assigned to a task only if it satisfies all function requirements of the task. Inequality (8) ensures that a resource can be assigned to a task only if it is installed on the appropriate branch of the resource hierarchy, on the location, station or line level. Function f is required at location l of station s if it is required by an assigned resource (9) or a task executed there (10). Constraint (11) states that every function requirement must be satisfied by a resource installed at or above the location of the requirement in the resource hierarchy. Inequality (12) accounts for the actual task durations, which might be greater than the duration values originally prescribed for the given combinations of task and resource due to synchronization constraints. Constraints (13) and (14) ensure that cycle time limits and the synchronizations are respected. Equation (15) defines the investment cost. Finally, lines (16), (17) and (18) list the binary variables in the model.

4. EXPLAINING THE SOLUTION

While computing an optimal solution to the model detailed in Section 3 might be mathematically straightforward using present-day MILP solvers, convincing the human expert about its suitability can pose a challenge. Doubts about the solution may arise due to the differences between the optimization model and the user's understanding of the problem, due to the lack of trust in the solution techniques, or it can happen that the input parameter values are not set in stone, but rather negotiable with other stakeholders in the overall design workflow.

In this paper, it is assumed that such concerns of the human expert are addressed during an interactive procedure, in which the expert asks consecutive "why not c" type questions about the solution. Here, in general, c can be

minimize
$$C \tag{1}$$
subject to
$$\sum_i y_{ti} = 1 \qquad \forall t \tag{2}$$
$$x_{i00} = 0 \qquad \forall i \notin \Lambda \tag{3}$$
$$x_{is0} = 0 \qquad \forall i \notin \Sigma \tag{4}$$
$$x_{isl} = 0 \qquad \forall i \notin \Pi \tag{5}$$
$$x_{i0l} = 0 \qquad \forall i, l \geq 1 \tag{6}$$
$$y_{ti} = 0 \qquad \forall t, i : \ R_t \not\subseteq P_i \tag{7}$$
$$y_{ti} \leq x_{is(t)l(t)} + x_{is(t)0} + x_{i00} \qquad \forall t, \ i \tag{8}$$
$$\rho_{fsl} \geq x_{isl} \qquad \forall i, s, l, f \in Q_i \tag{9}$$
$$\rho_{fs(t)l(t)} \geq 1 \qquad \forall t, f \in R_t \tag{10}$$
$$\rho_{fsl} \leq \sum_{i: f \in P_i} x_{isl} + x_{is0} + x_{i00} \qquad \forall f, s, l \tag{11}$$
$$\delta_t \geq \sum_i d_{ti} y_{ti} \qquad \forall t \tag{12}$$
$$T_j \geq \sum_{t: \ s(t)=s \ \wedge \ j(t)=j} \delta_t \qquad \forall j, s \tag{13}$$
$$\sum_{t \in \tau_1^k} \delta_t = \sum_{t \in \tau_2^k} \delta_t \qquad \forall k \tag{14}$$
$$C = \sum_{i,s,l} c_i x_{isl} \tag{15}$$
$$x_{isl} \in \{0, 1\} \qquad \forall i, s, l \tag{16}$$
$$y_{ti} \in \{0, 1\} \qquad \forall t, i \tag{17}$$
$$\rho_{fsl} \in \{0, 1\} \qquad \forall f, s, l \tag{18}$$

Fig. 1. MILP model of the line configuration problem.

any feature of a solution that can be encoded in a set of constraints on the decision variables in the MILP model. The actual prototype allows for the following types of questions:

- "Why don't we use resource i at location l of station s?": This feature can be enforced by adding side constraint $x_{isl} = 1$ to the MILP.
- "Why don't we use resource i for executing task t?": Side constraint $y_{ti} = 1$ is added.
- "Why cannot we do it at a lower cost of C'?": Inequality $C \leq C'$ is added.
- "Why cannot we achieve better productivity, reducing the cycle time of product j to T'_j?": The stricter upper bound on the cycle time is translated directly into an updated value of T_j.

The optimization model is then solved with the additional set of constraints translated from the user's questions. In case the restricted model is feasible, the answer to the user's question is "Yes, this is possible", and the detailed solution is presented. An interesting direction for further development is minimizing the difference between the updated solution and the original one. However, it is very unlikely to have several solutions with the same objective value in practical scenarios, and therefore, this direction is considered future research.

The main focus is the case when the model restricted by the user's constraints becomes infeasible. Then, an explanation of infeasibility consists of a set of constraints whose relaxation restores feasibility, e.g., "Your request can be satisfied if the cycle time of product 3 can be increased from 36 s to 40 s". To find such explanations, the constraints of the MILP model that can be relaxed are identified and handled as soft constraints. These involve constraints that are possibly not part of the user's model; that express a preference rather than a hard requirement; or whose coefficients can be negotiated. In the current MILP model (1)-(18), the following constraints are regarded as soft constraints:

- Constraint (2) expressing that each task is assigned to a resource. Additional binary variables $\alpha_t^{(2)}$ are created, where $\alpha_t^{(2)} = 1$ indicates that constraint (2) is relaxed for task t.
- Inequalities (11) encoding that function requirements are satisfied. A binary variable $\alpha_{fsl}^{(11)}$ is created for each f, s, and l.
- Constraints (13) on the cycle time for each product j and station s. Binaries $\alpha_{js}^{(13)}$ are added.
- Synchronization constraints (14). A binary variable $\alpha_k^{(14)}$ is added for each synchronization constraint k.
- Equality (15) defining the investment cost. A single variable $\alpha^{(15)}$ is added.

Next, an optimization model for computing a minimum-cardinality explanation is constructed by adding the above binary variables to the MILP, and replacing each soft constraint with the indicator constraint that states that the original constraint holds unless the assigned binary variable has a value of 1. E.g., the task assignment constraint (2) is replaced with the indicator constraint:

$$(\alpha_t^{(2)} = 0) \;\Rightarrow\; \sum_i y_{ti} = 1 \quad \forall t$$

Most state-of-the-art MILP solvers support such indicator constraints to express logical inference. In less advanced solvers, indicator constraints can be converted into linear inequalities with appropriate *big-M* coefficients.

Finally, the original objective function is replaced with the minimization of the number of violated soft constraints:

$$V = \sum_i \alpha_i^{(2)} + \sum_{fsl} \alpha_{fsl}^{(11)} + \sum_{js} \alpha_{js}^{(13)} + \sum_k \alpha_k^{(14)} + \alpha^{(15)}$$

Multiple alternative minimum-cardinality explanations may exist, and each of them might be of interest to the user. Two alternative search procedures were implemented to find all such explanations. Both procedures start with minimizing V. Then, the first procedure, called *Solution pooling* relies on the ability of the MILP solver to find *all* solutions with $V = V_{\min}$, and collect each of them into a solution pool. Two solutions are regarded as different if they differ in the values of the binary variables (including both the variables of the original MILP and the additional binaries).

In contrast, the *Iterative* procedure computes the alternative explanations one by one. Let A_m denote the set of binary variables assigned to the constraints relaxed in the explanation found in iteration m. Then, in iteration $m+1$, additional constraints of the form

$$\sum_{\alpha \in A_{m'}} \alpha \;\leq\; |A_{m'}| - 1 \quad \forall m' = 1, ..., m$$

are added to ensure that the new solution is different from any previous explanation.

Finally, the constructed explanations are presented to the user. To show how these explanations support the user in understanding the results of the optimization engine, the proposed techniques are illustrated on a medium-scale example in the Section 5.

5. ILLUSTRATIVE EXAMPLE

The proposed approach was validated on a medium-scale example motivated by the Bosch Rexroth i4.0 Mechatronics Training System, see Fig. 2. The line performs the assembly of a single product from two parts. The first, feeding station loads the two parts onto the conveyor and inspects them. The two parts are transferred separately to the second, assembly station, which results in a synchronization between the two stations. Finally, the assembled products are stored in the high-bay warehouse of the third station. The overall assembly process consists of 17 tasks, which require altogether 13 resources for their execution. While 10 out of the 13 resources are unambiguously determined by the function requirements, 3 resources can be selected from multiple candidates, resulting in different trade-offs between cycle time and investment cost.

The cheapest feasible configuration costs EUR 38 450, and incurs a cycle time of 39.6 seconds. In the example, the line designer wants to understand why this configuration just misses the target cycle time of 39 seconds. In the proposed approach, this translates to adding the constraints $C \leq 38\ 450$ and $T \leq 39$ (for convenience, the product

Fig. 2. The Bosch Rexroth i4.0 Mechatronics Training System.

index is omitted), resulting in an infeasible problem. The proposed approach generates the following 10 alternative explanations of infeasibility, each corresponding to the relaxation of a single soft constraint in the original line configuration model:

To achieve feasibility, the designer can...

- remove one of the given seven tasks from the process plan, i.e., relax constraint (2) for some $t \in \{5, 6, 7, 8, 13, 14, 15\}$. Indeed, these seven tasks define the single critical path in the schedule, see Fig. 3. Removing any of them decreases the cycle time to at most 38.2 seconds. Such a modification of the process plan may be feasible, e.g., inspection tasks can be performed offline, before loading the parts. Eliminating any other task does not improve performance.
- remove the synchronization between the first and the second stations, relaxing constraint (14). This can be achieved, e.g., by adding a buffer between the two stations.
- speed up the tasks in the second station to reduce their total duration by 0.6 seconds, i.e., relax constraint (13) for station 2. Note that the other two stations already satisfy the desired cycle time.
- Negotiate resource costs, relaxing constraint (15). This makes it possible to use more advanced and faster resources.

The benefit of these explanations for the user is that they help identify the most likely (i.e., minimal) difference between the optimization model and the user model. After conducting the necessary analysis, the user can either adjust the optimization model to align with the user model or accept the optimization model as valid and the computed solution as truly optimal. In either case, these explanations contribute to high-quality automated solutions also acceptable for the user.

The approach was implemented using the FICO Xpress v8.8 MILP solver, and experiments were run on a laptop computer with Intel i7 1.80 GHz CPU and 16 GB RAM.

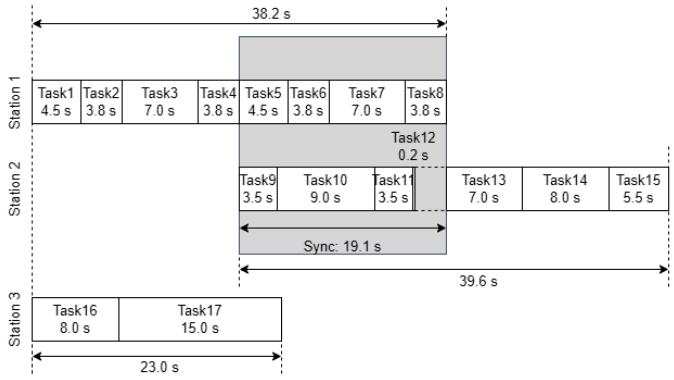

Fig. 3. Gantt chart of the tasks executed at the three stations.

The computation times for the two implemented search procedures, *Solution pooling* and *Iterative*, are displayed in Table 2. Finding the first explanation using either procedure required just slightly more computation time than solving the original problem (0.05 versus 0.04 seconds). Finding all 10 alternative explanations took 0.35–0.36 seconds. The significant difference in the performance of the two solvers became apparent when looking for a proof that no further explanations exist. With the simple Solution pooling approach, search did not terminate within the designated two-hour time frame. In contrast, the Iterative procedure could prove completeness immediately after finding those 10 explanations.

Table 2. Computation times.

	Solution pooling	Iterative
Original configuration problem: 0.04 s		
First explanation	0.05 s	0.05 s
All 10 explanations	0.35 s	0.36 s
Proof of completeness	> 2 h	0.36 s

6. CONCLUSIONS AND FUTURE RESEARCH

This paper proposes a method for computing explanations for production system configuration problems that provide valuable insight into why given solutions are proposed by automated design software. This is particularly relevant in the field of early-stage production system configuration, where a significant portion of the input parameters originate from other steps in a complex design workflow. Hence, explanations help identify the parameters and constraints that require special attention and can be negotiated with fellow experts working on the related design steps. Such decision support helps achieve goals that seem unrealizable based on the output of a classical optimization engine.

The reported results are just a first step towards explaining the solutions to such production system design problems, and some research questions are still open. Perhaps the main limitation of the current approach is that the number of alternative explanations can become huge. To tackle this issue, search must focus on constructing diverse explanations that are the most likely to identify the differences between the optimization model and the user model. Moreover, the computed explanations must be presented to the user in a comprehensible way, using graphical illustrations and natural language explanations, to contribute to the success of an iterative, mixed-initiative decision process.

REFERENCES

Aigner, K.M., Goerigk, M., Hartisch, M., Liers, F., and Miehlich, A. (2024). A framework for data-driven explainability in mathematical optimization. In *Proceedings of the 38th AAAI Conference on Artificial Intelligence*, volume 38, 20912–20920.

Battaïa, O., Dolgui, A., and Guschinsky, N. (2020). Optimal cost design of flow lines with reconfigurable machines for batch production. *International Journal of Production Research*, 58(10), 2937–2952.

Becker, C. and Scholl, A. (2006). A survey on problems and methods in generalized assembly line balancing. *European Journal of Operational Research*, 168(3), 694–715.

Boysen, N., Schulze, P., and Scholl, A. (2022). Assembly line balancing: What happened in the last fifteen years? *European Journal of Operational Research*, 301(3), 797–814.

Chakraborti, T., Sreedharan, S., Zhang, Y., and Kambhampati, S. (2017). Plan explanations as model reconciliation: moving beyond explanation as soliloquy. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI'17)*, 156–163.

Dobrovoczki, P., Kovács, A., Sakata, H., and Tsutsumi, D. (2024). Integrated system configuration and layout planning for flexible manufacturing systems. *Journal of Manufacturing Systems*, 77, 384–397.

Erwig, M. and Kumar, P. (2024). Explanations for combinatorial optimization problems. *Journal of Computer Languages*, 79, 101272.

Fox, M., Long, D., and Magazzeni, D. (2017). Explainable planning. In *Proceedings of the IJCAI-17 Workshop on Explainable AI*.

Goerigk, M. and Hartisch, M. (2023). A framework for inherently interpretable optimization models. *European Journal of Operational Research*, 310(3), 1312–1324.

ISO/IEC (2020). Software and systems engineering–software testing. Part 11: Guidelines on the testing of AI-based systems. Technical Report ISO/IEC TR 29119-11:2020.

Krarup, B., Coles, A., Long, D., and Smith, D.E. (2024). Explaining plan quality differences. In *Proceedings of the 34th International Conference on Automated Planning and Scheduling (ICAPS 2024)*, 324–332.

Krarup, B., Krivic, S., Magazzeni, D., Long, D., Cashmore, M., and Smith, D.E. (2021). Contrastive explanations of plans through model restrictions. *Journal of Artificial Intelligence Research*, 72, 533–612.

Linardatos, P., Papastefanopoulos, V., and Kotsiantis, S. (2021). Explainable AI: A review of machine learning interpretability methods. *Entropy*, 23(1).

Mersha, M., Lam, K., Wood, J., AlShami, A.K., and Kalita, J. (2024). Explainable artificial intelligence: A survey of needs, techniques, applications, and future direction. *Neurocomputing*, 599, 128111.

Nguyen, V., Vasileiou, S.L., Son, T.C., and Yeoh, W. (2020). Explainable planning using answer set programming. In *Proceedings of the 17th International Conference on Principles of Knowledge Representation and Reasoning (KR 2020)*, 662–666.

Ott, C. and Jäkel, F. (2024). Simplifex: Simplifying and explaining linear programs. *Cognitive Systems Research*, 88, 101298.

Tsutsumi, D., Kovács, A., and Szalóki, Á. (2022). Novel heuristic approach to integrating task sequencing and production system configuration. *Procedia CIRP*, 107, 28–33.