2026 European Conference on Computing in Construction
Corfu, Greece
July 12-15, 2026

GRAPH-BASED GENERATIVE SCHEDULING IN CONSTRUCTION PROJECT MANAGEMENT

Jianpeng Cao[1], Hisham Said[2]
[1]The University of Hong Kong, Hong Kong S.A.R., China
[2]Santa Clara University, Santa Clara, CA, United States

Abstract

Factory-based modular construction resembles manufacturing systems, but production scheduling remains challenging due to design variability, product heterogeneity, and design–production coupling. Existing construction scheduling methods rely on predefined task structures and adapt poorly to design changes. This paper presents a graph-based generative scheduling framework that derives production plans directly from modular designs. A design graph is transformed through graph rewrite rules to generate tasks, resources, precedence constraints, and task durations. The resulting job shop scheduling problem is solved using constraint programming. Case studies demonstrate automated schedule generation, reveal resource bottlenecks, and support early-stage production capacity planning.

Introduction

The increasing adoption of modular and prefabricated construction has intensified the need for advanced production planning and scheduling methods that can effectively manage complex task precedence, resource constraints, and design variability (Said and Kandimalla 2018). Factory-based construction environments share many characteristics with manufacturing systems, yet construction-specific uncertainties, heterogeneous products, and tight coupling between design and production continue to limit the effectiveness of traditional scheduling approaches (Ballard and Howell 1998; Gibb and Isack 2003). In particular, translating design information into executable production schedules remains a largely manual and error-prone process.

Job shop scheduling models have been widely applied to prefabrication and industrialized construction to optimize makespan, resource utilization, and workflow reliability. Specifically, job shop scheduling has been applied to the fabrication of pipe spools (ElMenshawy et al. 2025; Karim et al. 2024; Liu et al. 2017; Safarzadeh et al. 2018), precast concrete elements (Anvari et al. 2016; Xie et al. 2023), and sheet metal assemblies (Said and Kandimalla 2018). However, most existing approaches rely on predefined task structures and static process models, which restrict adaptability to design changes and limit scalability across different module configurations. Recent research has highlighted the potential of graph-based representations to model production systems more flexibly, capturing both structural and operational relationships in a unified framework.

This paper proposes a graph-based generative scheduling methodology that directly links modular design representations with production planning through graph rewrite rules. By transforming a parametric module design graph into a task and resource graph, the approach enables automated formulation of a job shop scheduling problem with design-dependent task durations and constraints. While the underlying optimization follows standard job shop scheduling (JSP) formulations, the innovation lies in the automated, graph-based generation of the constraint space, a process that traditionally relies on manual, error-prone expert input. In this context, the constraint space encompasses the set of logical dependencies (such as precedence, resource availability, and spatial conflicts) that bound the feasible solutions for a project. By serving as a generative engine, the proposed graph transformation process maps architectural topology directly onto these constraints. The framework is demonstrated through a modular production case study and solved using constraint programming to generate optimized schedules.

Literature review

Production planning and scheduling for prefabricated and modular construction has been extensively studied using job shop and flexible job shop formulations. Early and foundational work adapted manufacturing-oriented models to construction fabrication contexts, including steel components, bridge girders, precast elements, and pipe spools, often focusing on minimizing makespan and balancing resource utilization (Hasan et al. 2019; Hu et al. 2014; Liu et al. 2017). Genetic algorithms, memetic algorithms, and heuristic methods have been commonly employed to address the computational complexity of these problems (Meng et al. 2018; Moghadam et al. 2017; Xie et al. 2023).

More recent studies have expanded these models to incorporate multi-objective optimization, distributed production, and just-in-time constraints, reflecting the increasing complexity of industrialized construction systems (Anvari et al. 2016; Flamini and Nicosia 2024; Wang et al. 2024). Data-driven and learning-based approaches have also emerged, leveraging historical production data to forecast labor hours and improve scheduling performance (ElMenshawy et al. 2025; Karim et al. 2024; Said and Kandimalla 2018).

Graph-based representations have been recognized as a powerful abstraction for scheduling problems, particularly through disjunctive graph formulations that explicitly encode precedence and resource conflicts (Błażewicz et al. 2000; Otala et al. 2021). Besides, these scheduling advancements, an increasing number of

studies have explored graph-based modeling for building design in offsite and modular construction, particularly due to its effectiveness in representing the physical topology and spatial relationships of prefabricated modules (Cao et al. 2024; Gan 2022). While these approaches offer strong theoretical foundations for both design and production independently, their application in construction has largely focused on modeling production processes rather than integrating upstream design information. Consequently, the linkage between modular design attributes and schedule generation remains weak.

The present study addresses this gap by employing graph rewriting to generatively derive task, resource, and constraint structures from a design graph, enabling automated, design-consistent job shop scheduling for modular construction.

Methodology

Graph Representation

The design of a prefabricated building module design is modeled as an undirected graph in which nodes represent rooms and walls. Room nodes indicate spaces with geometric attributes (width, length, area), while wall nodes are attributed by one of three types (internal, external, or partition wall) and are annotated with the wall's dimensional properties (height and length) and the presence of doors or windows. The edges between room and wall nodes capture boundary relationships, indicating which walls define the perimeter of each room. On the other hand, the edges between room nodes indicate the adjacency and accessibility between two rooms without the existence of a partition wall. This graph-based representation captures efficiently the room–wall topology of a module design that can be informative for production planning.

Rule Definition

This framework relies on a rule repository that bridges architectural design and production planning. To ensure the scalability of the generative approach, the framework categorizes rules into a multi-level hierarchy.

Element Level: At this granular level, rules specify the tasks and associated resources for individual components (e.g., interior walls, MEP fixtures). This ensures that every physical part of the parametric design triggers a corresponding manufacturing requirement.

Room Level: These rules capture the aggregated activities required for specific functional spaces, such as dry rooms versus wet rooms. This level introduces broader finishing tasks, including ceiling installation, flooring, and specialized commissioning.

Module Level: At the highest level of the hierarchy, the rules introduce an abstract "module node." This node associates the entire unit with holistic tasks such as structural assembly, final quality inspection, and logistics.

By organizing rules in this manner, the framework effectively deconstructs the fabricator's production expertise into a reusable Rule Library. While the initial formulation of these rules requires input from the fabricator crew to ensure the "constraint space" accurately reflects the reality of the shop floor, the resulting templates can be applied to new designs with minimal intervention. This reduces the user's burden from "creating logic" to "selecting appropriate rule sets," thereby enhancing the system's usability for non-expert project managers.

Design-to-Production Transformation using Graph Rewriting Rules

Graph rewriting, also known as graph transformation, is a rule-based approach for modifying graph structures by adding, removing, or reconfiguring nodes and edges according to predefined rules. In engineering design, it provides a structured yet flexible mechanism for automating complex transformations, encoding domain knowledge, and systematically exploring design alternatives (Kolbeck et al. 2022). Recent advances have demonstrated its effectiveness in modular and parametric design contexts, where rule-based transformations enable automated generation of design variants and support early-stage decision-making (Aksamija et al. 2010; Voss et al. 2023). Graph grammars extend this concept by offering a formal framework for design synthesis, allowing valid configurations to be generated through well-defined transformation rules while ensuring consistency and correctness through metamodels or type graphs (Campbell 2009; Schmidt and Rudolph 2016).

For the proposed design-production framework, graph rewriting rules are used to link process knowledge, including task, resources and constraints, to design graph. As such, there are two production-related nodes that are added to the design graph by the graph rewriting rules: 1) task nodes represent a discrete production activity with uninterrupted duration; and 2) resource nodes represent the production crews or equipment used to accomplish one task or more. Task assignment to each module design element (wall and room nodes) is created by adding edges "constructs" from task nodes to the design elements. Resource assignment is created by adding edges "uses" from task nodes to resource nodes representing required production capabilities. Afterwards, constraint assignment is created by adding edges "precedes" between task nodes to represent the logical sequencing between the fabrication tasks.

Graph Rule Assignment

Rule assignment is performed by iteratively matching the left-hand side (LHS) patterns of the graph rewrite rules against the module design graph. When a valid matching is found, the corresponding rule is applied by augmenting the graph with the right-hand side (RHS) components. The RHS introduces task nodes, resource dependency links, and task–task precedence relations, while preserving the original design entities and their topological relationships.

After the rule assignment, task durations are computed by associating the unit of the design entity with a predefined unit production rate stored in the task node. For example, the duration of the task "Light-gauge wall framing (studs

& door/window frames, exterior)" is calculated by multiplying the wall surface area extracted from the design graph by the corresponding production rate (hours per square meter of wall). This approach ensures the task durations are design-dependent and adapt to the design changes automatically, ensuring the consistency between design and production planning.

Scheduling Formulation

After the task assignment, the production planning is formulated as a job shop scheduling problem, where each module to be produced is modeled as a job. The input is the graph generated from the graph rewrite process, which defines the set of tasks, their precedence relationships, required resources, and design-dependent durations. Besides, two additional inputs, including the number of modules to be produced and the number of stations (resources), are defined according to order demand and factory production constraints. In traditional job shop models, stations are fixed in location. However, this proposed framework models them as moveable resources (crews) as most prefabricated volumetric modules are stationary and the fabrication crews move between them to execute the tasks. The scheduling objective is to minimize the overall makespan of the module production process.

Implementation

The proposed methodology is implemented as a Python-based computational pipeline that integrates graph modeling, task graph generation, and constraint-based scheduling. To demonstrate the methodology, a sample module design is used as a case study (Figure 1), which was obtained from a large building fabricator in Asia. In total, the analyzed module includes three exterior walls (EW), four interior walls (IW), two interior wet walls (WW), one dry room (R), and two wet rooms (WR). The differentiation between EW, IW, and WW wall elements is needed due to their different compositions and layers (partition material, insulation, existence of mechanical and electrical M&E systems, and finish material. A similar justification exists for the differentiation between dry and wet rooms (R and WR).

Wall Data

Wall	Dimension	Doors	Windows
EW1	3 x 2.12	-	1
EW2	3 x 2.51	-	1
EW3	3 x 4.63	-	2
IW1	3 x 6.88	1	-
IW2	3 x 6.88	-	-
IW3	3 x 4.10	-	-
IW4	3 x 4.10	1	-
WW1	3 x 2.12	-	-
WW2	3 x 4.10	-	-

Room Data

Room	Area
R1	4.63 x 6.88
WR1	2.12 x 4.10
WR2	2.51 x 4.10

EW = Exterior Wall
IW = Interior Wall
WW = Wet Wall
R = Room
WR = Wet Room

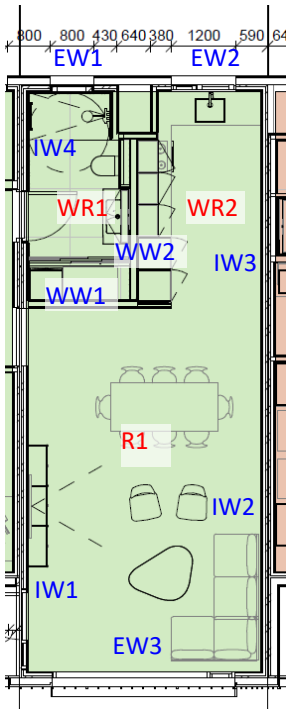

Figure 1: A sample module design

In addition to the prefabricated module design, it was critical to formulate a product-process recipe for each element in the module to facilitate the generative schedule process. Table 1 lists the product-process recipes for each element and aspect of a module, including production tasks, station/resource, unit type, duration, and precedence constraints. These recipes reflect the fabrication of the overall structural system of the module (structural steel post-beam with a concrete floor) and the fabrication tasks and resources needed to realize the different wall and room types. The recipes were formulated by the researcher by examining the production plan that were shared by the building fabricator. It should be noted that the fabricator shared a generic schedule for all their standard modules, which was useful to strategically outline the production process and resources but this generic schedule did not enable reliable scheduling and control of the production process. This observation confirms the need for this study and the proposed framework for generative scheduling of prefabricated building modules.

Table 1. Module production process

Element	ID	Production Task	Station / Resource	Unit	Duration (hrs)	Preceding Tasks
Strcutural	1	Construct the module structural steel frame & concrete for	Structural	Module	24	-
	2	Construct the module concrete floor	Structural	Module	8	1
	3	Strip the concrete formwork	Structural	Module	4	2, lag 32 hours
Exterior Wall	4	Light-gauge wall framing (studs & door/window frames, ext	Framing	m^2, wall	0.53	2, lag 16 hours
	5	Wall electrical installation	M&E	m^2, wall	0.27	21
	6	Insulation installation	Insulation	m^2, wall	0.20	5
	7	Interior gypsum board installation	Framing	m^2, wall	0.27	6
	8	Exterior water proofing	Insulation	m^2, wall	0.27	4
	9	Window & door installation and testing	Doors & Windows	Count, D&W	1.00	7,8
	10	Exterior galvanized panel installation and sealing	Framing	m^2, wall	0.53	9
Interior Regular Wall	11	Light-gauge wall framing (studs & door/window frames)	Framing	m^2, wall	2.00	21
	12	Wall electrical installation	M&E	m^2, wall	0.75	11
	13	Interior gypsum board installation	Framing	m^2, wall	0.50	12
	14	Door installation	Doors & Windows	Count, D	1.00	13
Interior Wet Wall	15	Light-gauge wall framing (studs & door/window frames)	Framing	m^2, wall	1.50	29
	16	Wall electrical installation	M&E	m^2, wall	0.75	15
	17	Wall plumbing installation	M&E	m^2, wall	1.25	15
	18	Interior gypsum board installation	Framing	m^2, wall	0.50	16, 17
	19	Wall waterproofing	Insulation	m^2, wall	0.20	18
	20	Door installation	Doors & Windows	Count, D	1.00	19
Dry Room	21	ceiling, Install mechanical and electrical	M&E	m^2, room	0.50	4
	22	ceiling, Light-gauge framing	Framing	m^2, room	2.00	21
	23	ceiling, Fireproofing	Insulation	m^2, room	0.25	22
	24	ceiling, install gypsum board	Framing	m^2, room	0.75	23
	25	Interior wall & ceiling mudding & taping	Painting	m^2, room+w	0.27	13, 24
	26	Vinyl flooring installation	Flooring	m^2, room	0.13	25
	27	Interior wall & ceiling painting	Painting	m^2, room+w	0.08	14, 26
	28	Power commissioning	M&E	room	2.00	12, 21
Wet Room	29	ceiling, Install mechanical and electrical	M&E	m^2, room	0.60	4
	30	Floor drain installation	M&E	room	3.00	29
	31	Floor waterproofing	Insulation	m^2, room	0.89	19, 30
	32	Tiling for floor and walls	Flooring	m^2, room+w	0.13	20, 31
	33	Bathroom flood testing	Insulation	room	4.00	32
	34	Cabinet installation	Carpentry	room	4.00	33
	35	Water fixture installation & commissioning	M&E	room	4.00	34
	36	Power commissioning	M&E	room	2.00	16, 29
Closeout & Logistics	37	Inspection	Closeout	Module	8.00	10, 28, 34, 36
	38	Module sealing and shippig	Closeout	Module	8.00	37

The implementation leverages *ReGraph* and *NetworkX* as the core graph modeling and transformation engines. *NetworkX* ("NetworkX" 2026) is used to construct and manage the underlying design and task graphs, providing a flexible data structure for representing nodes, edges, and associated attributes such as geometry, resources, and production parameters. *ReGraph* ("ReGraph" 2026) is employed to define and execute graph rewrite rules, enabling rule-based pattern matching and transformation of the design graph into a task and resource graph. Through left-hand-side pattern matching and right-hand-side graph augmentation, *ReGraph* supports the systematic addition of tasks, precedence relations, and resource dependencies while preserving the original design topology. This combination allows the implementation to separate graph data management from transformation logic, ensuring that design-dependent task generation is modular, extensible, and computationally efficient within the Python-based pipeline.

The task graph is then converted into a job shop scheduling problem and solved using Google OR-Tools CP-SAT ("OR-tools" 2026). The objective of the experimental evaluation is to analyze how the generative framework supports production planning through three key performance metrics: makespan, busy time, station saturation, and queue time. For the initial analysis, the production of a single module is examined under the assumption of deterministic task durations and fixed station capacities. Production station capacities are assumed based on typical factory conditions, including three framing stations, two M&E stations, two insulation stations, one doors-and-windows station, two painting stations, two flooring stations, one carpentry station, two structural workstations, and one closeout station.

Results

This section presents the outputs generated by the proposed framework. Figure 2 shows the graph representation of the sample module design. Rooms and walls are modeled as nodes, with edges capturing their topological relationships. Wall nodes store geometric properties such as dimensions and the number of doors and windows, and room nodes store the width, length and area. These geometric attributes are used to derive the duration calculation for each task.

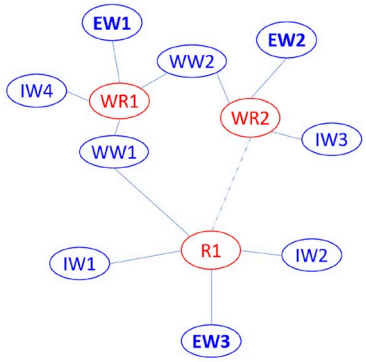

Figure 2: Graph representation of sample module design

Built upon the product-process recipes (Table 1), graph rewrite rules are applied to transform the design graph into a task graph, as illustrated in Figure 3. The resulting task graph represents design entities (blue circular nodes), production tasks (green squares) and resources/stations (orange diamonds). Solid edges represent task–entity and task–resource associations, while dashed edges indicate precedence constraints between tasks.

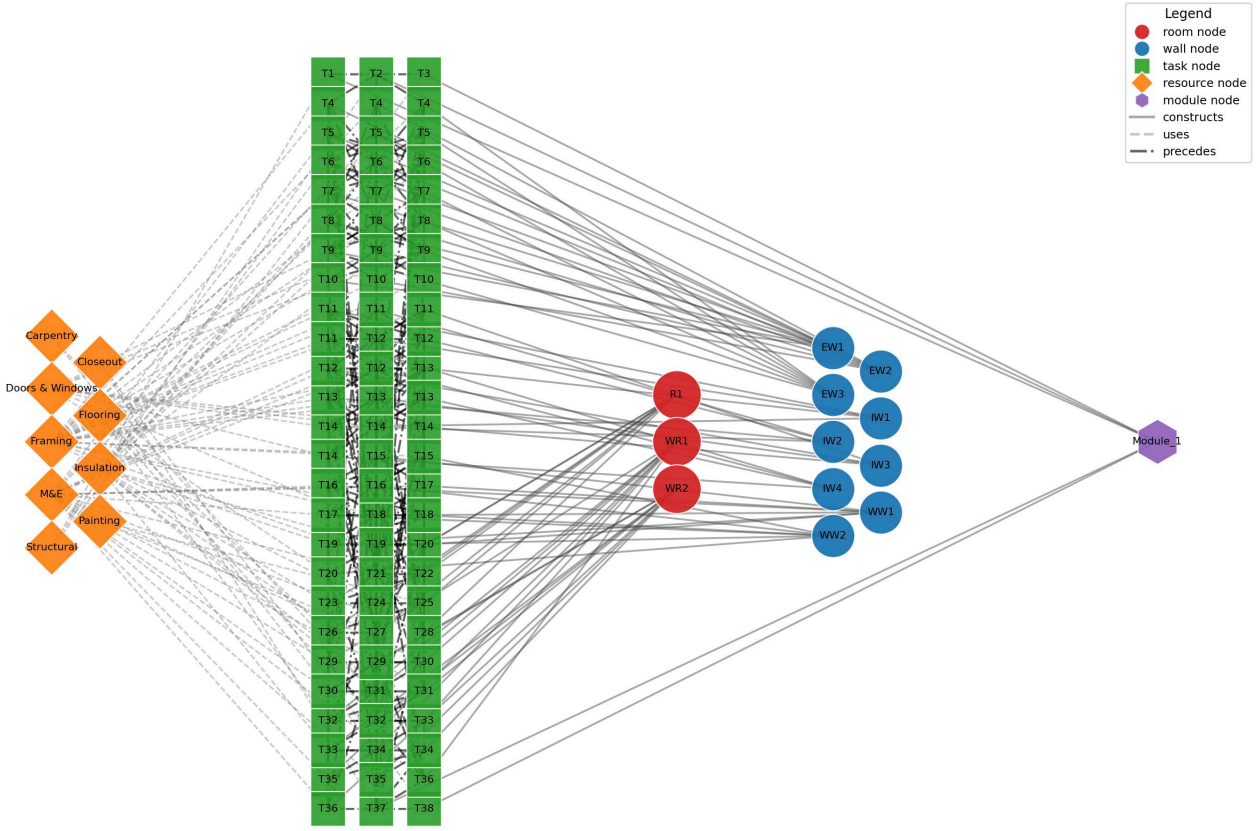

Figure 3: Task graph representation

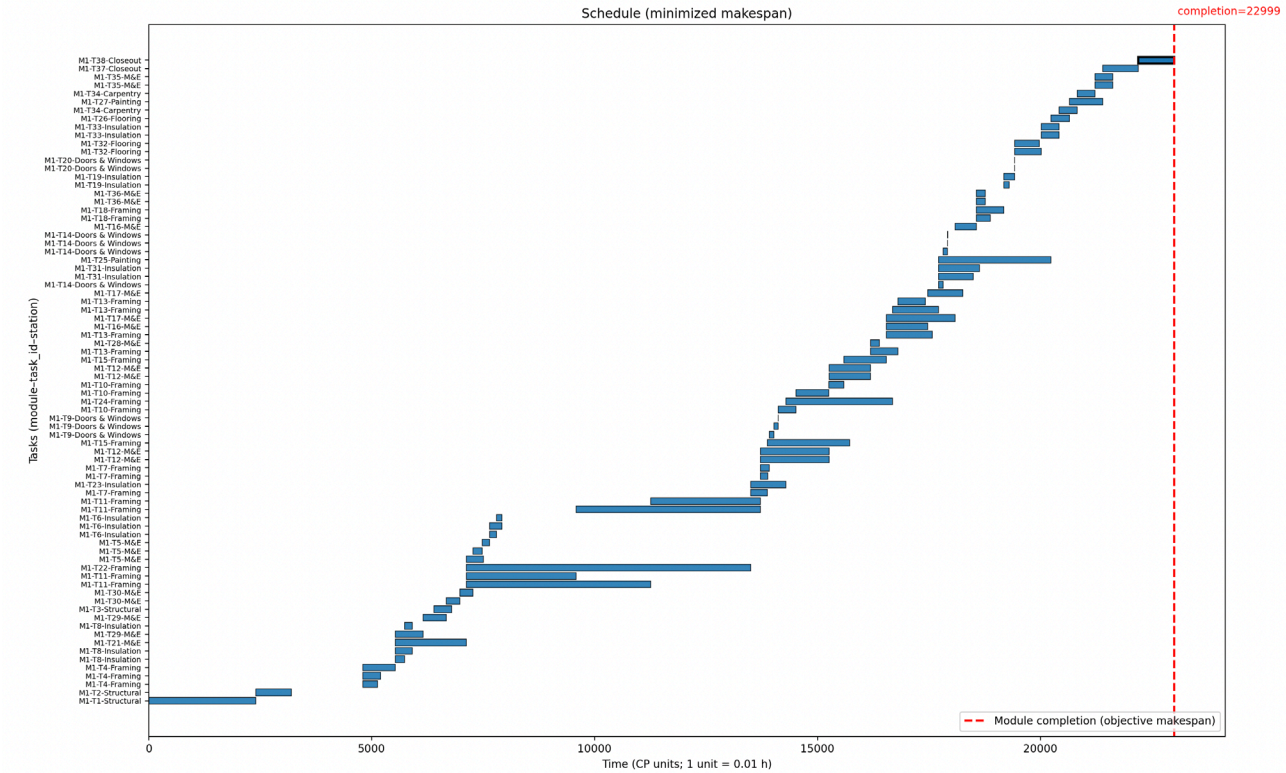

Figure 4: Optimized job shop schedule for modular production

The optimized schedule is presented as a Gantt chart (Figure 4) obtained by solving the job shop scheduling problem derived from the task graph. The vertical axis lists individual tasks, while each bar represents the corresponding task start and finish times. The dashed vertical line indicates the optimized makespan (230 hours), which corresponds to the module completion time.

Discussion

The results demonstrate the ability of the proposed graph-based generative scheduling framework to automatically translate a parametric modular design into a feasible and optimized production schedule. Beyond the single-module demonstration, additional experiments were conducted to derive important insights for the system scalability, resource bottlenecks, and the interaction between design-driven task generation and factory production constraints.

In contrast, professionals are enabled by the proposed generative framework to be more efficient by automating the mapping of architectural topology to scheduling constraints, the system eliminates manual "linking" errors, ensuring that the generated schedule is logically consistent and adheres strictly to design dependencies.

Expanding the analysis beyond the scheduling of a single module revealed the scalability limitation and the diminishing marginal efficiency of the modeled production system. Table 2 shows the resulting total makespan and the average module makespan for fabricating different numbers of modules. The multi-module experiments reveal a clear non-linear relationship between production volume and total makespan. While increasing the number of modules leads to economies of repetition in early stages, the average completion time per module eventually increases as shared resources become saturated. This diminishing marginal efficiency is a well-known phenomenon in prefabrication environments, where parallelism is constrained by finite station capacities and reentrant workflows. The observed reduction in average module time from three to approximately twelve modules indicates improved utilization of parallel stations, whereas the subsequent increase reflects congestion effects and resource contention. Importantly, these dynamics emerge naturally from the generated task and resource graph without manual adjustment of precedence logic or capacity rules. This highlights a key advantage of the proposed approach over traditional scheduling models that rely on static, predefined workflows. Because task durations and precedence relationships are derived directly from the design graph and rule assignments, the scheduling model remains consistent as production volume changes, allowing emergent system behavior to be analyzed rather than imposed.

Table 2: The resulting total makespan and average module makespan under different number of modules

Number of Modules	Total Makespan (hr)	Average Module Makespan (hr)
1	230.00	230.00
3	413.65	137.88
5	634.23	126.85
10	1,180.79	118.08
11	1,293.16	117.56
12	1,402.08	116.84
13	1,650.35	126.95
14	1,802.5	128.75
15	1,960.00	130.67
20	2,655.12	132.76

The station-level performance analysis further illustrates the explanatory power of the graph-based formulation. The framework was utilized to analyze the resource utilization and bottlenecks under the scenario of fabricating 20 modules. Three metrics were tracked for each station: 1) busy time: measures total processing workload executed on station; 2) saturation percentage: represents the fraction of time during which the station operates at full capacity, indicating how frequently the station constrains task execution; and 3) queue time: measures the cumulative waiting time of tasks assigned to a station before processing, capturing upstream congestion effects. Tables 3 lists the values of these metrics and the capacities for three sample stations: framing, mechanical and electrical, and insulation. Framing stations exhibited the highest busy time and saturation percentage, confirming their role as the primary bottleneck in the examined production configuration. This finding is consistent with practical observations in modular factories, where framing activities often dominate early production stages and constrain downstream flow. In contrast, insulation and finishing-related stations showed relatively low saturation, suggesting underutilized capacity and potential opportunities for rebalancing or task reallocation. Queue time metrics provided additional insight into upstream congestion effects that are difficult to capture using aggregate performance indicators alone. High queue times at framing and M&E stations indicate that delays propagate through the system not merely due to task durations, but because of structural dependencies encoded in the task graph. From a managerial perspective, this suggests that capacity expansion decisions should prioritize stations with both high saturation and high queue accumulation, rather than focusing solely on busy time.

Table 3: The station performance analysis results

Station	Capacity	Busy Time	Saturation %	Queue time
Framing	3	6531.2	81.3	255570.4
M&E	2	2830.6	53.0	46339.9
Insulation	2	993.0	17.6	6247.52

Conclusion

This study illustrates how graph rewriting can serve as an effective bridge between design representation and production scheduling. By separating design topology, process knowledge, and scheduling constraints into distinct but interoperable graph layers, the approach maintains both flexibility and interpretability. The use of constraint programming further ensures that the generated schedules are globally optimized with respect to the defined objective, rather than relying on heuristic sequencing alone. More broadly, the results suggest that graph-based generative scheduling can function not only as a planning tool, but also as an analytical instrument for understanding the systemic consequences of design and capacity decisions in industrialized construction.

A key contribution of this study lies in demonstrating how tight coupling between design and production can be operationalized rather than abstracted away. Because task generation and duration calculation are driven by geometric attributes extracted from the design graph, even minor design changes, such as wall lengths, room configurations, or the number of openings—would propagate automatically into the production schedule. This capability contrasts with most existing job shop scheduling approaches in construction, which assume fixed task sets and rely on manual updates when designs evolve. The experimental results suggest that such coupling is particularly valuable in early-stage design evaluation and production planning. Designers and production planners can use the framework to explore "what-if" scenarios, assessing how design decisions influence makespan, station saturation, and bottleneck severity before committing to detailed fabrication plans. In this sense, the framework supports a shift from reactive scheduling toward proactive, design-informed production optimization.

Nevertheless, the current implementation also has limitations. The experiments assume deterministic task durations and station capacities, whereas real-world factory environments are subject to stochastic disruptions, learning effects, and workforce variability. In addition, the case study focuses on a single module type, and while the framework is generalizable, heterogeneous module families and mixed-model production were not explicitly examined. These factors may amplify congestion effects and introduce additional scheduling complexity.

The findings from the extended experiments point to several promising avenues for future research. First, incorporating stochastic task durations and probabilistic disruptions would allow the framework to support robustness and resilience analysis. Second, extending the graph rewrite rules to handle multiple module typologies within a single production line would enable mixed-model scheduling studies. Third, coupling the scheduling outputs with cost, labor, or energy metrics could support multi-objective optimization and decision-making at the factory planning level. Fourth, the framework successfully transforms a parametric module design into a feasible schedule without manual intervention. Future work can leverage this capability to identify design features that have the greatest impact on production performance.

References

Aksamija, A., K. Yue, H. Kim, F. Grobler, and R. Krishnamurti. 2010. "Integration of knowledge-based and generative systems for building characterization and prediction." Artificial Intelligence for Engineering Design, Analysis and Manufacturing, 24 (1): 3–16. https://doi.org/10.1017/S0890060409990138.

Anvari, B., P. Angeloudis, and W. Y. Ochieng. 2016. "A multi-objective GA-based optimisation for holistic Manufacturing, transportation and Assembly of precast construction." Automation in Construction, 71: 226–241. https://doi.org/10.1016/j.autcon.2016.08.007.

Ballard, G., and G. Howell. 1998. "What Kind of Production Is Construction?"

Błażewicz, J., E. Pesch, and M. Sterna. 2000. "The disjunctive graph machine representation of the job shop scheduling problem." European Journal of Operational Research, 127 (2): 317–331. https://doi.org/10.1016/S0377-2217(99)00486-5.

Campbell, M. 2009. "A Graph Grammar Methodology for Generative Systems." 25.

Cao, J., H. Said, A. Savov, and D. Hall. 2024. "Graph-Based Evolutionary Search for Optimal Hybrid Modularization of Building Construction Projects." Journal of Construction Engineering and Management, 150 (8): 04024098. American Society of Civil Engineers. https://doi.org/10.1061/JCEMD4.COENG-14687.

ElMenshawy, M., L. Wu, B. Gue, and S. AbouRizk. 2025. "Automating Pipe Spool Fabrication Shop Scheduling for Modularized Industrial Construction Projects Using Reinforcement Learning." Journal of Computing in Civil Engineering, 39 (3): 04025013. American Society of Civil Engineers. https://doi.org/10.1061/JCCEE5.CPENG-6042.

Flamini, M., and G. Nicosia. 2024. "A generalized disjunctive graph model for a complex production problem." Procedia Computer Science, International Conference on Industry Sciences and Computer Science Innovation, 237: 289–296. https://doi.org/10.1016/j.procs.2024.05.107.

Gan, V. J. L. 2022. "BIM-based graph data model for automatic generative design of modular buildings."

Automation in Construction, 134: 104062. https://doi.org/10.1016/j.autcon.2021.104062.

Gibb, A., and F. Isack. 2003. “Re-engineering through pre-assembly: client expectations and drivers.” Building Research & Information, 31 (2): 146–160. Routledge. https://doi.org/10.1080/09613210302000.

Hasan, M., M. Lu, and K. Bird. 2019. Planning and scheduling bridge girders fabrication through shop-floor operations simulation.

Hu, X., M. Lu, and S. AbouRizk. 2014. “BIM-based data mining approach to estimating job man-hour requirements in structural steel fabrication.” In: Proc. Winter Simul. Conf. 2014, 3399–3410.

Karim, M. R., B. Gue, L. Wu, and Y. Mohamed. 2024. “Job Labour-Hours Forecasting for a Pipe Spool Fabrication Shop Using Data Mining.” In: Proc. Can. Soc. Civ. Eng. Annu. Conf. 2023 Vol. 5, edited by S. Desjardins, G. J. Poitras, and M. Nik-Bakht, 437–449. Cham: Springer Nature Switzerland.

Kolbeck, L., S. Vilgertshofer, J. Abualdenien, and A. Borrmann. 2022. “Graph Rewriting Techniques in Engineering Design.” Frontiers in Built Environment, 7: 815153. https://doi.org/10.3389/fbuil.2021.815153.

Liu, J., M. Soleimanifar, and M. Lu. 2017. “Resource-loaded piping spool fabrication scheduling: material-supply-driven optimization.” Visualization in Engineering, 5 (1): 5. https://doi.org/10.1186/s40327-017-0044-3.

Meng, R., Y. Rao, Y. Zheng, and D. Qi. 2018. “Modelling and solving algorithm for two-stage scheduling of construction component manufacturing with machining and welding process.” International Journal of Production Research, 56 (19): 6378–6390. https://doi.org/10.1080/00207543.2017.1349949.

Moghadam, A., K. Y. Wong, and H. Piroozfard. 2017. “Solving a hybrid jobshop scheduling problem with space constraints and reentrant processesby using an improved hybrid genetic algorithm: A case study.” International Journal of Industrial Engineering : Theory Applications and Practice, 24: 483–504.

“NnetworkX.” 2026. Python. NetworkX.

“OR-tools.” 2026. C++. Google.

Otala, J., A. Minard, G. Madraki, S. Mousavian, J. Otala, A. Minard, G. Madraki, and S. Mousavian. 2021. “Graph-Based Modeling in Shop Scheduling Problems: Review and Extensions.” Applied Sciences, 11 (11). publisher. https://doi.org/10.3390/app11114741.

“ReGraph.” 2026. Python. Kappa-Dev.

Safarzadeh, S., S. Shadrokh, and A. Salehian. 2018. “A heuristic scheduling method for the pipe-spool fabrication process.” Journal of Ambient Intelligence and Humanized Computing, 9 (6): 1901–1918. https://doi.org/10.1007/s12652-018-0737-z.

Said, H. M., and P. Kandimalla. 2018. “Performance Measurement of Building Sheet-Metal Ductwork Prefabrication under Batch Production Settings.” Journal of Construction Engineering and Management, 144 (2): 04017107. American Society of Civil Engineers. https://doi.org/10.1061/(ASCE)CO.1943-7862.0001423.

Schmidt, J., and S. Rudolph. 2016. “Graph-Based Design Languages: A Lingua Franca for Product Design Including Abstract Geometry.” IEEE Computer Graphics and Applications, 36 (5): 88–93. https://doi.org/10.1109/MCG.2016.89.

Voss, C., F. Petzold, and S. Rudolph. 2023. “Graph transformation in engineering design: an overview of the last decade.” AI EDAM, 37: e5. Cambridge University Press. https://doi.org/10.1017/S089006042200018X.

Wang, S., X. Li, L. Gao, and J. Li. 2024. “A multi-disjunctive-graph model-based memetic algorithm for the distributed job shop scheduling problem.” Advanced Engineering Informatics, 60: 102401. https://doi.org/10.1016/j.aei.2024.102401.

Xie, Y., H. Wang, G. Liu, and H. Lu. 2023. “Just-in-Time Precast Production Scheduling Using Dominance Rule-Based Genetic Algorithm.” IEEE Transactions on Neural Networks and Learning Systems, 34 (9): 5283–5297. https://doi.org/10.1109/TNNLS.2022.3217318.