

An Improved Adaptive and Dynamic Constraint Satisfaction Model for Large-Scale Scheduling

¹H. N. Akpan, ^{*2}N. D. Nwiabu, ^{*3}D. J. S. Sako

^{1,2,3}Department of Computer Science, Rivers State University, Port-Harcourt, Nigeria

E-mail: ¹akpan.happiness@rsu.edu.ng, ²nwiabu.nuka@ust.edu.ng, ³dum.sako@ust.edu.ng

Abstract: Large-scale dynamic scheduling becomes computationally expensive when resource changes cause the complete constraint graph to be reconsidered even though only a limited part of the schedule is affected. This paper develops and evaluates an improved Adaptive and Dynamic Constraint Satisfaction Problem (A-DCSP) model that combines graph partitioning, impact localization, incremental repair, localized constraint propagation, and controlled boundary synchronization. University course timetabling was represented as a constraint graph, while lecturer- and room-unavailability events were localized through bounded graph neighbourhoods. Assignments outside the affected region remained fixed during repair, and cross-partition constraints incident to the region were checked before global schedule validation. The model was implemented in Python with Google OR-Tools and evaluated against global recomputation and global incremental propagation using the ITC-2007 comp01, comp05, and comp07 benchmark instances and deterministic synthetic workloads containing 100–2,000 courses. The evaluation comprised 486 real-benchmark observations, 108 partition-sensitivity observations, and 810 synthetic observations. On the real benchmarks, the partitioned A-DCSP reduced mean update latency by 70.6%–96.6%, inspected 68.6%–96.3% fewer variables, checked 54.6%–95.1% fewer constraints, and produced 99.1%–99.8% fewer assignment changes than global recomputation. At 2,000 synthetic courses, it inspected 4.611 variables and 23.500 constraints, with a mean update latency of 0.6422 seconds. All 1,404 observations recorded zero hard-constraint violations. The results show that event-dependent graph localization reduces update cost and schedule disruption while controlled boundary coordination maintains feasibility within a centralized software environment.

Keywords: Dynamic scheduling, constraint satisfaction, graph partitioning, impact localization, incremental repair.

I. INTRODUCTION

Constraint satisfaction provides a formal basis for scheduling tasks whose assignments are restricted by resource, temporal, and conflict relationships. University course timetabling is a representative application in which lectures must be assigned to periods and rooms without lecturer, curriculum, or room conflicts [1]. The problem becomes dynamic when a lecturer or room becomes unavailable after a feasible timetable has been produced.

Complete reconstruction after every event can increase update time and alter assignments that remain valid. Incremental repair preserves unaffected assignments [2], [3], but a graph-wide incremental method may still inspect the complete constraint structure. Incremental scheduling therefore benefits from a repair scope determined by the event rather than by the total problem size [4].

Graph partitioning can organize a large constraint graph into smaller regions, while impact localization identifies the

vertices reached from an event. A localized repair must still check constraints that cross from affected to unaffected partitions. Dynamic distributed constraint research establishes the need for coordinated adaptation [5], and communication-aware search shows that coordination activity should be measured explicitly [6]. However, these studies do not provide a centralized timetable model that combines event-localized repair with selective boundary coordination.

This paper develops and evaluates an improved Adaptive and Dynamic Constraint Satisfaction Problem (A-DCSP) model for large-scale scheduling. The model combines graph partitioning, bounded impact localization, incremental repair, localized propagation, and controlled boundary synchronization. It is evaluated against global recomputation and global incremental propagation using three ITC-2007 instances and deterministic synthetic workloads of 100–2,000 courses. The evaluation focuses on update latency, variables inspected, constraints checked, assignment changes, hard-constraint violations, and boundary-coordination activity.

II. RELATED WORKS

Dynamic scheduling methods revise an existing plan after job arrival, machine failure, maintenance, or another resource event. Schedule instability has been evaluated alongside makespan, flow time, and workload, confirming that the number of changes introduced during repair is important [8]. Event-driven and periodic rescheduling have also been compared under orders, due-date changes, breakdowns, cancellations, and urgent jobs [9]. Hybrid metaheuristic methods extend this work to multiple breakdown and arrival scenarios [10]. These methods support responsive rescheduling, but their local operations refer to movements in the solution-search space rather than to an event-defined constraint-graph region.

University timetable recovery is more directly concerned with retaining valid assignments. Robust timetabling has measured the changes and quality loss required to recover from single and multiple disruptions [2]. Recovery experiments on 21 ITC-2007 instances reported an average minimum of 4.8 timetable perturbations, a median repair-iteration time of 50 seconds, and a mean of 794 seconds [3]. These studies establish repair latency and schedule stability as relevant measures, but they do not report the variables, constraints, or partition boundaries processed during an update.

Timetabling research has reduced complexity through decomposition. Network-flow formulations reduce the number of integer variables while preserving curriculum relationships [11]. Other work classifies subproblems according to whether teachers, rooms, or periods are assigned first [12], while mixed-integer formulations use reduction and local improvement to manage joint student, room, and period assignment [13]. These approaches exploit structure during timetable construction, but their subproblems are fixed by the formulation rather than identified from a dynamic event.

Distributed constraint optimization addresses coordination when connected decisions are divided among problem regions. Complete inference has been made more scalable by propagating several smaller utility tables instead of one joint table [14]. Proactive dynamic optimization represents possible future changes [5], while communication-aware local search evaluates message latency and loss [6]. This literature shows that decomposition cannot ignore coordination, but agent-to-agent messaging is not equivalent to boundary checking within a centralized timetable graph.

Incremental ordering reuses information from an existing schedule rather than restarting after modification [4], and a two-

level CSP framework demonstrates configurable constraint processing for operational scheduling and routing problems containing hundreds of activities [7]. The unresolved issue is whether event-dependent graph localization can reduce update latency, processing scope, and schedule disruption while relevant cross-partition checks preserve global feasibility.

III. SYSTEM DESIGN

The A-DCSP model combines a persistent scheduling state with an event-dependent update path. The persistent state contains the scheduling problem, current feasible timetable, global constraint graph, and graph-partition map. The update path applies a lecturer- or room-unavailability event, identifies the affected graph region, fixes assignments outside that region, repairs the movable assignments, checks relevant cross-partition constraints, and validates the resulting complete timetable. Figure 1 presents the components and information flow of this architecture.

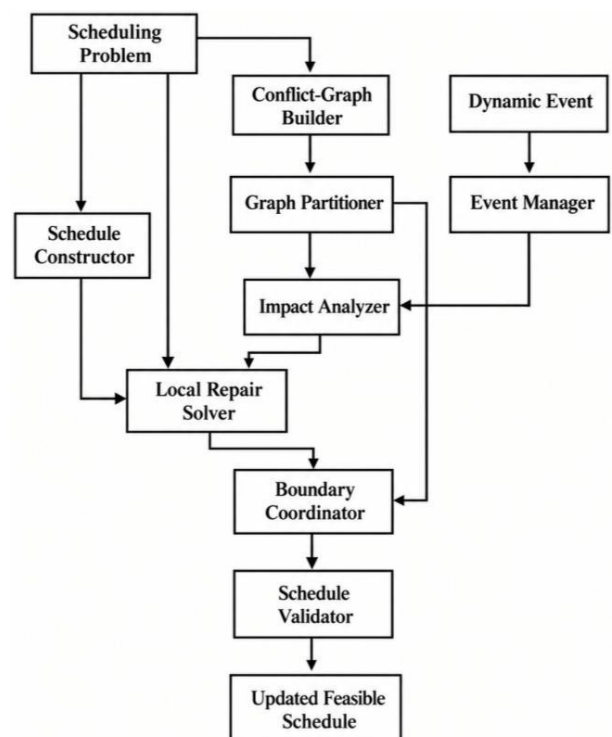


Figure 1: Architecture of the partitioned A-DCSP scheduling model

The scheduling-problem input supplies the Schedule Constructor and Conflict-Graph Builder, while the dynamic-event input is processed by the Event Manager. The Graph Partitioner organizes the conflict graph into regions without removing cross-partition edges. Event seeds pass to the Impact

Analyzer, which determines the repair scope. The Local Repair Solver modifies assignments within that scope, and the Boundary Coordinator checks cross-partition relationships incident to the affected variables. The Schedule Validator then checks the complete repaired timetable and releases an updated feasible schedule. This organization preserves the same input and output boundary as a global-update scheduler while replacing complete-graph update processing with localized repair and controlled boundary coordination.

Evaluation used the comp01, comp05, and comp07 instances from the ITC-2007 Curriculum-Based Course Timetabling collection. The ECTT records supplied courses, weekly lecture counts, lecturers, rooms, periods, curricula, course-time restrictions, and course-room restrictions. Each weekly lecture occurrence was expanded into a separate scheduling variable. Curriculum membership and shared lecturers created conflict edges between lecture variables. A

complete assignment associated every lecture variable with one time slot and one room. Feasibility required one assignment per lecture, exclusive room occupancy, exclusive lecturer occupancy, separation of conflict-adjacent lectures, and compliance with time, room, lecturer-availability, and room-availability restrictions. Room capacity remained a soft condition for the ITC-2007 instances in accordance with their original formulation.

Synthetic workloads extended the evaluation to 100, 250, 500, 1,000, and 2,000 courses. Each workload contained 40 time slots, eight department identifiers, room capacities from 40 to 120, and seed-generated enrolments and lecturer assignments. For n courses, the number of lecturers was $\max(10, \lfloor n/4 \rfloor)$ and the room count was $\lceil (n/40) \times 2.5 \rceil$. Shared lecturers generated structural conflicts, and room capacity was enforced as a hard condition. Table 1 summarizes the real benchmark instances after expansion into lecture variables.

Table 1: Characteristics of the ITC-2007 benchmark instances

Instance	Courses	Lecture variables	Lecturers	Rooms	Time slots	Curricula	Curriculum-conflict edges
comp01	30	160	24	6	30	14	1,682
comp05	54	152	47	9	36	139	2,491
comp07	131	434	99	20	25	77	4,208

The global constraint graph is denoted by $G = (V, E)$, where V contains all lecture variables and E contains curriculum and shared-lecturer conflicts. The deterministic graph-partitioning procedure selects the highest-degree unassigned vertex as a seed and grows a region through adjacent unassigned vertices ordered by degree. For m partitions, the vertex set is divided into non-overlapping regions as

$$V = \bigcup_{i=1}^m V_i, \quad V_i \cap V_j = \emptyset \quad (i \neq j). \quad (1)$$

The initial target size of each region is $\lceil |V|/m \rceil$, with the limit recalculated from the unassigned vertices and remaining partitions during construction. Lexical identifier order resolves equal-degree ties, making the partition map repeatable for a fixed graph and partition count. Cross-partition edges remain in the global graph for later boundary checks.

A lecturer-unavailability event adds a lecturer-slot restriction and uses every lecture taught by that lecturer as an event seed. A room-unavailability event adds a room-slot restriction and uses the lecture occupying that room and slot as its seed. From the seed set S_δ of event δ , bounded breadth-first traversal identifies every vertex whose shortest graph distance from a seed does not exceed radius r . The affected set is

$$A_r(\delta) = \{v \in V : \min_{s \in S_\delta} \text{mindist}_G(s, v) \leq r\}. \quad (2)$$

Only variables in $A_r(\delta)$ are movable. Every variable outside the set is fixed to its current slot-room assignment. For benchmark repair, CP-SAT creates variables for permitted lecture-slot-room combinations and enforces the complete hard-feasibility rules. Among feasible updated schedules $\mathcal{R}(\delta)$, the objective minimizes changes within the affected set while preserving all assignments outside it:

$$a^* = \arg \min_{a' \in \mathcal{R}(\delta)} \sum_{v \in A_r(\delta)} \mathbf{1}[a'_v \neq a_v], a'_v = a_v \quad \forall v \notin A_r(\delta). \quad (3)$$

The synthetic experiments used a common deterministic constructor and repair procedure for all three update methods. Existing assignments were retained when feasible; invalid assignments were replaced by testing permitted slot-room pairs in a fixed order. This control separated update-scope behaviour from differences between solving technologies.

Boundary coordination operates on the retained global graph. If $\pi(v)$ is the partition label of vertex v , the relevant cross-partition boundary edges for affected set A are

$$C_b(A) = \{(u, v) \in E: \pi(u) \neq \pi(v) \wedge (u \in A \vee v \in A)\}. \quad (4)$$

The contacted partitions are $P_A = \{\pi(v): v \in A\}$. With P_{S_δ} denoting the partitions containing the event seeds, synchronization activity is recorded as

$$Y = |P_A \setminus P_{S_\delta}|. \quad (5)$$

The structural constraint count provides a solver-independent measure of processing scope. For scope Q , it comprises three unary feasibility categories per variable and every unique graph edge incident to the scope:

$$H(Q) = 3|Q| + |\{(u, v) \in E: u \in Q \vee v \in Q\}|. \quad (6)$$

Global recomputation and global incremental propagation use $Q = V$, whereas partitioned A-DCSP uses $Q = A_r(\delta)$. Global recomputation constructs a complete schedule after the event without retaining current assignments. Global incremental propagation may retain valid assignments but still inspects the complete graph. Partitioned A-DCSP fixes unaffected assignments, repairs the localized set, and checks relevant cross-partition edges before complete validation.

The benchmark experiment used three instances, seeds 42, 43, and 44, three repetitions per seed, six cumulative events per method, and three methods, producing 486 event observations. Corresponding methods received independent copies of the same initial schedule and ordered event sequence. CP-SAT used a 120-second time limit and eight search workers. Partitioned A-DCSP used four partitions and radius one. A sensitivity experiment varied the partition count over 2, 4, and 8 and the radius over 1 and 2 on comp01, producing 108 observations. The synthetic experiment used five workload sizes, three seeds, three repetitions, six events, and three methods, producing 810 observations; eight partitions and radius one were used for the partitioned method.

Experiments ran under Windows 11 Pro 64-bit on an Intel Core i7-2620M at 2.70 GHz with four logical processors and 11.95 GB RAM. The software environment comprised Python 3.10.0, Google OR-Tools 9.14.6206, NumPy, pandas, and Matplotlib. Update latency was measured from immediately before event application until repair, boundary processing, and global validation were complete. Additional measures were variables inspected, structural constraints checked, changed assignments, hard-constraint violations, boundary checks, contacted partitions, synchronization operations, partition imbalance, and edge cut. Raw event observations were retained before aggregation, and every initial and updated schedule passed through the same hard-constraint validator. The experiment matrix fixed the instances, methods, seeds, repetitions, event count, solver settings, partition configurations, localization radii, and synthetic sizes before execution.

IV. RESULTS AND DISCUSSION

Execution used Windows 11 Pro, an Intel Core i7-2620M processor, 11.95 GB RAM, Python 3.10.0, and OR-Tools 9.14.6206. The benchmark experiment fixed three seeds, three repetitions, six cumulative events, a 120-second CP-SAT limit, and eight search workers. The benchmark, sensitivity, and synthetic experiments produced 486, 108, and 810 observations, respectively.

The principal comparison covered update latency, propagation scope, schedule stability, feasibility, and boundary coordination. The three methods processed corresponding initial schedules and event sequences. Table 2 presents the main ITC-2007 results, with 54 observations in each instance-method row.

Table 2: Main performance results on the ITC-2007 instances

Instance	Method	Mean latency (s)	Variables inspected	Constraints checked	Assignments changed	Hard violations
comp01	Global recomputation	12.696	160.000	2,294.000	158.593	0
comp01	Global incremental	13.507	160.000	2,294.000	1.500	0
comp01	Partitioned A-DCSP	2.518	39.000	706.000	1.500	0
comp05	Global recomputation	19.947	152.000	2,998.000	147.593	0
comp05	Global incremental	19.100	152.000	2,998.000	1.111	0
comp05	Partitioned A-DCSP	5.858	47.667	1,361.167	1.111	0
comp07	Global recomputation	66.057	434.000	5,813.000	432.389	0
comp07	Global incremental	62.427	434.000	5,813.000	1.000	0
comp07	Partitioned A-DCSP	2.232	16.000	284.500	1.000	0

4.1 Update Efficiency

Partitioned A-DCSP recorded the lowest mean update latency on all three benchmark instances. Relative to global recomputation, latency decreased by 80.2% on comp01, 70.6% on comp05, and 96.6% on comp07. Relative to global incremental propagation, the reductions were 81.4%, 69.3%, and 96.4%. Figure 2 presents the benchmark latency results and their 95% confidence intervals.

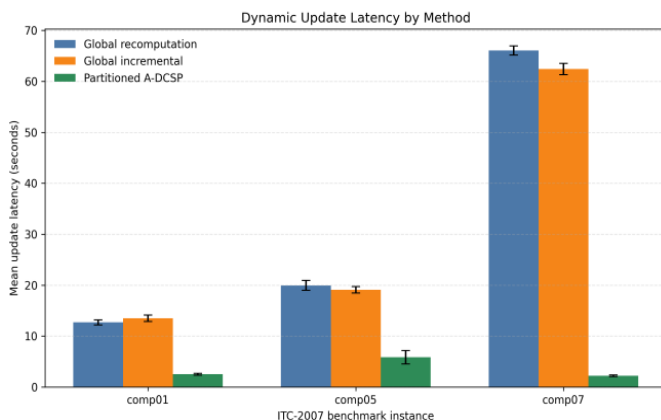


Figure 1: Mean dynamic-update latency on the ITC-2007 benchmark instances

The global methods inspected every lecture variable, whereas partitioned A-DCSP inspected means of 39.000, 47.667, and 16.000 variables on comp01, comp05, and comp07. These represent reductions of 75.6%, 68.6%, and 96.3%. Constraint checks decreased by 69.2%, 54.6%, and 95.1%, respectively. Figure 3 shows the difference in propagation scope.

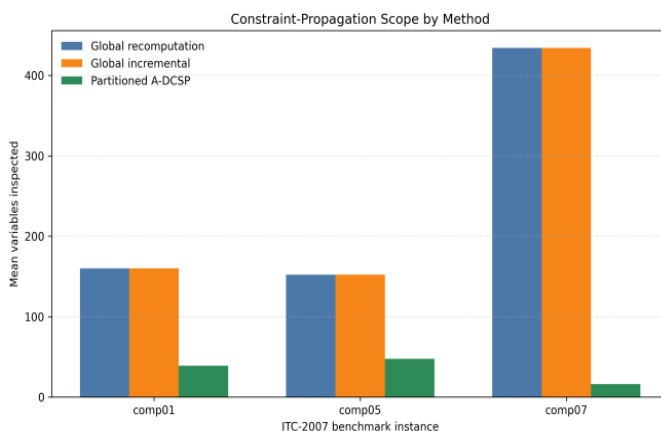


Figure 2: Mean number of variables inspected during benchmark updates

Comp07 was the largest benchmark but produced the smallest affected scope and lowest partitioned latency. Event location, graph connectivity, and incident boundary structure therefore influenced localized update cost more directly than total instance size.

4.2 Schedule Stability and Boundary Coordination

Global recomputation changed means of 158.593, 147.593, and 432.389 assignments. Global incremental propagation and partitioned A-DCSP changed identical means of 1.500, 1.111, and 1.000 assignments. Partitioned repair produced 99.1%–99.8% fewer changes than global recomputation while matching global incremental stability. Its additional advantage was obtaining that stability with less graph inspection and lower latency, consistent with disruption-recovery research [2], [3].

Mean boundary checks were 172.000 on comp01, 748.833 on comp05, and 34.500 on comp07. Corresponding synchronization means were 1.000, 2.167, and 0.500, while contacted partitions averaged 2.000, 3.333, and 1.500. Comp05 combined the greatest boundary activity with the highest partitioned latency. Sensitivity analysis showed that increasing the impact radius from one to two expanded mean repair scope from 39 to 63 variables and raised latency from approximately 3.0 to 5.2 seconds. At eight partitions, the wider radius contacted five partitions and recorded four synchronization operations.

All 1,404 observations recorded zero hard-constraint violations. The smaller repair scope therefore preserved feasibility when combined with relevant boundary checks and complete post-repair validation.

4.3 Synthetic Scalability

At 2,000 synthetic courses, both global methods inspected 2,000 variables and checked 9,934 constraints, while partitioned A-DCSP inspected 4.611 variables and checked 23.500 constraints. These values represent reductions of approximately 99.77% and 99.76%.

Mean latency at 2,000 courses was 1.1060 seconds for global recomputation, 2.1406 seconds for global incremental propagation, and 0.6422 seconds for partitioned A-DCSP. Across the five sizes, partitioned latency was 40.4%–51.5% lower than global recomputation and 63.8%–70.0% lower than global incremental propagation. Its affected scope remained between 4.611 and 5.722 variables as workload size increased from 100 to 2,000 courses.

The evaluation is limited to a centralized CPU-based prototype, three ITC-2007 instances, two availability-event types, and synthetic workloads of at most 2,000 courses. Synchronization represents software-level boundary activity rather than network time or data volume. Distributed deployment and other scheduling domains require separate evaluation.

V. CONCLUSION

This paper developed and evaluated an improved A-DCSP model that combines graph partitioning, bounded impact localization, incremental repair, localized constraint propagation, and controlled boundary synchronization for large-scale dynamic scheduling. On the three ITC-2007 instances, partitioned A-DCSP reduced mean update latency by 70.6%–96.6%, inspected 68.6%–96.3% fewer variables, and checked 54.6%–95.1% fewer structural constraints than global recomputation. It changed 99.1%–99.8% fewer assignments while matching the stability of global incremental propagation. At 2,000 synthetic courses, the model inspected 4,611 variables and 23,500 constraints and completed updates in 0.6422 seconds, compared with 1.1060 seconds for global recomputation and 2.1406 seconds for global incremental propagation. Boundary coordination contacted means of 1.500–3.333 partitions and recorded 0.500–2.167 synchronization operations per real-benchmark update. All 1,404 benchmark, sensitivity, and synthetic observations recorded zero hard-constraint violations.

The results show that restricting repair to the event-dependent graph neighbourhood can reduce update cost and unnecessary timetable disruption while relevant cross-partition checks preserve complete-schedule feasibility. The model is applicable to centralized scheduling environments in which resource events affect limited connected regions and existing assignments should be retained. Its evaluation is confined to university timetabling, lecturer- and room-unavailability events, software-level coordination measures, and workloads of at most 2,000 synthetic courses on one CPU-based computer. Future work should evaluate additional scheduling domains and event types, incorporate soft-constraint quality into dynamic repair, adapt partition count and impact radius to graph structure, and measure synchronization time, message volume, memory use, and speedup under distributed or parallel execution.

REFERENCES

- [1] Chen, M. C., Sze, S. N., Goh, S. L., Sabar, N. R., & Kendall, G. (2021). A survey of university course timetabling problem: Perspectives, trends and opportunities. *IEEE Access*, 9, 106515–106529.
- [2] Gülcü, A., & Akkan, C. (2020). Robust university course timetabling problem subject to single and multiple disruptions. *European Journal of Operational Research*, 283(2), 630–646.
- [3] Lindahl, M., Stidsen, T., & Sørensen, M. (2019). Quality recovering of university timetables. *European Journal of Operational Research*, 276(2), 422–435.
- [4] Neubert, S., & Casel, K. (2024). Incremental ordering for scheduling problems. In *Proceedings of the International Conference on Automated Planning and Scheduling*, 34, 405–413.
- [5] Hoang, K. D., Fioretto, F., Hou, P., Yeoh, W., Yokoo, M., & Zivan, R. (2022). Proactive dynamic distributed constraint optimization problems. *Journal of Artificial Intelligence Research*, 74, 179–225.
- [6] Rachmut, B., Zivan, R., & Yeoh, W. (2022). Communication-aware local search for distributed constraint optimization. *Journal of Artificial Intelligence Research*, 75, 637–675.
- [7] Zhang, L., Maqrot, S., Mouysset, F., & Bortolaso, C. (2025). A constraint satisfaction problems based scalable framework to address large-scale realistic scheduling and routing problems. In *Proceedings of the 14th International Conference on Operations Research and Enterprise Systems*, 45–56.
- [8] Gao, K., Yang, F., Zhou, M., Pan, Q., & Suganthan, P. N. (2018). Flexible job-shop rescheduling for new job insertion by using discrete Jaya algorithm. *IEEE Transactions on Cybernetics*, 49(5), 1944–1955.
- [9] Baykasoğlu, A., Madenoğlu, F. S., & Hamzadayı, A. (2020). Greedy randomized adaptive search for dynamic flexible job-shop scheduling. *Journal of Manufacturing Systems*, 56, 425–451.
- [10] Fuladi, S. K., & Kim, C. S. (2024). Dynamic events in the flexible job-shop scheduling problem: Rescheduling with a hybrid metaheuristic algorithm. *Algorithms*, 17(4), 142.
- [11] Bagger, N. C. F., Kristiansen, S., Sørensen, M., & Stidsen, T. R. (2019). Flow formulations for curriculum-based course timetabling. *Annals of Operations Research*, 280(1–2), 121–150.
- [12] Herres, B., & Schmitz, H. (2021). Decomposition of university course timetabling. *Annals of Operations Research*, 302(2), 405–423.
- [13] Rappos, E., Thiémarc, E., Robert, S., & Hêche, J. F. (2022). A mixed-integer programming approach for solving university course timetabling problems. *Journal of Scheduling*, 25(4), 391–404.



- [14] Deng, Y., Chen, Z., Chen, D., Zhang, W., & Jiang, X. (2019). AsymDPOP: Complete inference for asymmetric distributed constraint optimization problems. *In*

Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, 223–230.

Citation of this Article:

H. N. Akpan, N. D. Nwiabu, & D. J. S. Sako. (2026). An Improved Adaptive and Dynamic Constraint Satisfaction Model for Large-Scale Scheduling. *Journal of Artificial Intelligence and Emerging Technologies (JAJET)*. 3(10), 7-13. Article DOI: <https://doi.org/10.47001/JAIET/2026.310002>

*** End of the Article ***