STM JOURNALS

International Journal of
Data Structure Studies

ISSN: 3049-2440
Volume 4, Issue 1, 2026
January–June
DOI (Journal): 10.37591/IJDSS

https://journals.stmjournals.com/ijdss

Review IJDSS

Data Structure Driven Probabilistic Deadlock Resolution in Multiprocessor Systems

Kakinada Mahitha Sri Ishwarya[1], Chegondi Revathi[2], Manas Kumar Yogi[3,*]

Abstract

Deadlock resolution in multiprocessor systems is fundamentally a graph-theoretic and probabilistic decision problem. Existing victim selection heuristics, such as youngest, oldest, and lowest priority, apply static rules that overlook the dynamic runtime state of processes, leading to unnecessary computational loss. This paper reframes the inference-guided preemption (IGP) algorithm as a data-structure-centric solution, highlighting how resource allocation graphs, wait-for graphs, adjacency lists, min-heaps, and hash-based evidence stores interact to enable efficient deadlock detection and cost-minimized victim selection via Bayesian inference. A probabilistic cost model over three latent factors, such as remaining execution time, resource holding cost, and process criticality, drives a min-heap selection mechanism that selects the optimal victim in O(n log n). Simulation experiments using SimPy demonstrate a 34% reduction in resolution overhead relative to the best baseline heuristic, with comparable fairness and significantly faster workload shift adaptation, validating the structural and probabilistic design choices.

Keywords: Resource allocation graph, wait-for graph, Bayesian inference, deadlock resolution, multiprocessor systems, priority queue, process scheduling

INTRODUCTION

Deadlock is a fundamental problem in concurrent computing: a set of processes becomes permanently blocked because each process holds a resource that another process needs, forming a circular dependency [1, 2]. In multiprocessor environments, where dozens of processes share heterogeneous resources simultaneously, the probability of deadlock increases, and its cost, such as wasted CPU cycles, memory locks, and degraded throughput, grows proportionally.

Classical operating system textbooks present deadlock management through four pillars: prevention, avoidance, detection, and recovery [3]. Recovery by resource preemption is the most pragmatic strategy in high-concurrency environments: the operating system selects one process as a "victim," rolls it back, and redistributes its resources to break the deadlock cycle. The efficiency of this approach depends entirely on how the victim is selected.

Traditional victim selection heuristics, such as youngest, oldest, and lowest priority, are computationally inexpensive but structurally blind. They rely on the scalar attributes of individual processes without consulting the broader dependency graph or the probabilistic cost of disrupting a process mid-execution [4, 5]. This leads to suboptimal preemption choices that waste disproportionate CPU work time.

***Author for Correspondence**
Manas Kumar Yogi
E-mail: manas.yogi@gmail.com

[1,2]Student, Department of Computer Science and Engineering, Pragati Engineering College (A), Surampalem, Andhra Pradesh, India
[3]Assistant Professor, Department of Computer Science and Engineering, Pragati Engineering College (A), Surampalem, Andhra Pradesh, India

Received Date: March 29, 2025
Accepted Date: April 04, 2025
Published Date: April 28, 2026

Citation: Kakinada Mahitha Sri Ishwarya, Chegondi Revathi, Manas Kumar Yogi. Data Structure Driven Probabilistic Deadlock Resolution in Multiprocessor Systems. International Journal of Data Structure Studies. 2026; 4(1): 11–20p.

This paper reframes the inference-guided preemption (IGP) algorithm through a data structure lens. We show that the entire lifecycle of deadlock detection, evidence collection, probabilistic inference, and victim selection maps naturally onto a pipeline of well-understood data structures: directed graphs for dependency modelling, adjacency lists for traversal, hash maps for evidence storage, probability distribution objects for Bayesian updating, and min-heaps for cost-ordered victim extraction. By making these structural choices explicit, we expose the engineering foundations that make IGP both accurate and implementable in real operating system kernels.

The remainder of this paper is organized as follows. Section 2 describes the data structures used for deadlock modelling. Section 3 presents the Bayesian probabilistic framework and its structural representation. Section 4 details the design of the IGP algorithm. Section 5 analyzes computational complexity. Section 6 presents the evaluation results. Finally, Section 7 concludes the paper and provides directions for future work.

DATA STRUCTURES FOR DEADLOCK MODELLING

Resource Allocation Graph

The resource allocation graph (RAG) is a canonical data structure for modelling deadlock states [5]. A directed bipartite graph $G = (P \cup R, E)$, where P is the set of processes, R is the set of resource types, and E consists of two disjoint edge classes:

- Assignment edges $R \rightarrow P$: resource r has been allocated to process p.
- Request edges $P \rightarrow R$: process p is waiting to acquire resource r.

The graph is stored as an adjacency list, which is memory-efficient for sparse resource graphs typical in operating systems (where the number of edges $|E| << |P| \cdot |R|$). Each resource node additionally maintains a semaphore-count field indicating the number of available instances; multi-instance resources require a more general edge-weight representation.

A deadlock is equivalent to the existence of a cycle in RAG for single-instance resources or a more complex condition (involving reachability sets) for multi-instance resources. Cycle detection is performed using depth-first search (DFS) in $O(V + E)$ time, where $V = |P| + |R|$ and E is the total edge count [2].

Wait-for Graph

When all resource nodes are projected out of the RAG, the resulting structure is the wait-for graph (WFG)—a directed graph over processes only, where a directed edge $p \rightarrow q$ indicates that p is waiting for a resource currently held by q. The WFG is strictly smaller ($O(|P|)$ vertices) and admits faster cycle detection during runtime deadlock polling [3, 6].

In practice, the operating system maintains the WFG as a dynamic adjacency list that is updated for every resource allocation and release event. The cycle detection DFS marks nodes as WHITE (unvisited), GREY (on the current stack), or BLACK (fully explored). A back edge to a GREY node signals a deadlock cycle; the cycle members from the candidate set D passed to IGP for victim selection.

Comparative Analysis of Representation Choices

Table 1 summarizes the data structures employed in the deadlock detection and resolution pipeline, their structural types, roles, and asymptotic complexities.

The adjacency matrix representation offers $O(1)$ edge queries but incurs $O(V^2)$ space, which is prohibitive when hundreds of processes coexist with tens of resource types. The adjacency list is therefore preferred for both RAG and WFG. The min-heap is central to IGP victim selection; the hash map underpins evidence collection (Section 3). Both are standard library components in modern operating system implementations (e.g., Linux red-black trees and hash tables in the kernel).

Table 1. Data structures used in the deadlock detection and resolution pipeline.

Data structure	Type	Role in deadlock detection	Time complexity
Resource allocation graph (RAG)	Directed bipartite graph	Models' assignment and request edges between processes and resources	O(V + E)
Wait-for graph (WFG)	Directed graph	Detects circular wait conditions among processes only	O(V + E)
Adjacency matrix	2D array	Fast O(1) edge lookup; high memory use for sparse graphs	$O(V^2)$
Adjacency list	Array of linked lists	Memory-efficient traversal for sparse resource graphs	O(V + E)
Min-heap (priority queue)	Binary heap	Efficient extraction of minimum-cost victim in O(log n)	O(log n)
Hash map (evidence store)	Key-value store	O(1) storage and retrieval of per-process evidence vectors	O(1) avg.

BAYESIAN PROBABILISTIC FRAMEWORK AND STRUCTURAL REPRESENTATION

Probability Distributions as First-Class Data Structures

The IGP algorithm treats the cost of preempting a process as a latent variable, which is a quantity that cannot be observed directly but can be estimated from observable system signals. Bayesian inference provides an update rule. Each cost factor C_j is modelled as a continuous probability distribution object parameterized by (μ, σ^2) under Gaussian conjugacy [7]:

$$P(C_j \mid \text{evidence}) \propto P(\text{evidence} \mid C_j) \cdot P(C_j)$$

When both the prior and likelihood are Gaussian, the posterior remains Gaussian with closed-form parameter updates [8]. This conjugate structure ensures that each Bayesian update is a simple arithmetic operation on two floating-point parameters—mean and variance—rather than a numerical integration, making the approach suitable for real-time operating system scheduling loops.

Therefore, each process in the deadlock set D is associated with a tuple of three distribution objects:

- Distribution over remaining execution time (C_{rt})—parameterized by historical CPU usage logs.
- Distribution over resource holding cost (C_{rc})—parameterized by resource type weights and exclusivity.
- Distribution over criticality penalty (C_{cp})—parameterized by process priority and dependency fan-in.

Evidence Vectors and Hash Map Storage

When a deadlock is detected, the system constructs an evidence vector for each process $p \in D$. This vector is a fixed-size, typed record stored in a hash map keyed by the process identifier, supporting O(1) average-time read and write access:

```
EvidenceMap: HashMap<PID, EvidenceVector>

EvidenceVector {
  cpu_time_used   : float         // cumulative CPU seconds
  held_resources  : List<RID> // resource identifiers
  in_cycle_degree : int          // WFG indegree in deadlock subgraph
  priority        : int          // static scheduling priority
  process_type    : Enum{IO, CPU, TRANS}
}
```

The hash map is populated in a single O(n) scan of the kernel process table, where n = |D|. Because deadlock cycles in practice involve small process counts (typically 2–8 processes), this scan is negligible relative to the overall scheduling overhead.

Bayes' Theorem Applied to Victim Cost Estimation

For each process $p \in D$ and each cost factor $C_j \in \{C_{rt}, C_{rc}, C_{cp}\}$, the system performs a Bayesian update using the relevant evidence. For the remaining time C_{rt}, the update uses *cpu_time_used* as a proxy: higher CPU consumption suggests a lower remaining time for I/O-bound processes but a higher remaining time for CPU-bound processes, calibrated by the *process_type* field.

The posterior expected value $E[C_j \mid \text{evidence}]$ is computed analytically using the Gaussian conjugate model. The total expected preemption cost for process p is then a weighted combination:

$$E[\text{Cost}(p)] = w_{rt} \cdot E[C_{rt} \mid e] + w_{rc} \cdot E[C_{rc} \mid e] + w_{cp} \cdot E[C_{cp} \mid e]$$

Where, weights w_{rt}, w_{rc}, and $w_{cp} \in [0,1]$ sum to unity and can be adjusted per system policy (throughput-optimized vs. criticality-aware). The expected value of a Gaussian posterior is simply its updated mean; therefore, the computation reduces to three multiplications and two additions per process—constant time regardless of the distribution complexity [9].

THE IGP ALGORITHM: DATA STRUCTURE DESIGN

Cost Model Architecture

The three cost factors are mapped directly to observable kernel metrics, forming a structured cost model. Each factor is represented as a parameterized Gaussian object stored in a per-process CostModel record:

```
CostModel {
  prior_rt   : GaussianDist // prior on remaining time
  prior_rc   : GaussianDist // prior on resource cost
  prior_cp : GaussianDist // prior on criticality
  w_rt, w_rc, w_cp : float // policy weights
}
```

CostModel records are maintained in a secondary hash map indexed by the process class (IO-bound, CPU-bound, and transactional), allowing prior distributions to be initialized from class-level offline statistics rather than individual process histories during early system operation [10].

Min-Heap for Victim Selection

After computing E[Cost(p)] for all $p \in D$, the IGP algorithm selects the process with the minimum value. This is achieved using a min-heap (binary heap) built over the computed values. For $|D| = n$ processes, insertion takes O(log n) per element, and the extraction of the minimum takes O(log n). Therefore, the total victim selection step runs in O(n log n), a marginal increase over the O(n) linear scan of static heuristics, justified by the significant reduction in preemption overhead [11].

Algorithm Pseudocode

The complete IGP procedure is summarized below:

```
Algorithm IGP(D: Set<Process>) → victim: Process
  Input: D = {p1, p2,..., pn} // deadlock cycle members
  Output: victim process with minimum expected cost

  evidenceMap ← new HashMap<PID, EvidenceVector>
  costHeap ← new MinHeap<(float, PID)>

  // Step 1: Collect evidence from the kernel process table
  for each p in D:
  evidenceMap[p.pid] ← collect_evidence(p)
```



```
// Step 2: Bayesian update and cost estimation
for each p in D:
e ← evidenceMap[p.pid]
post_rt ← bayesian_update(costModel[p.type].prior_rt, e.cpu_time_used)
post_rc ← bayesian_update(costModel[p.type].prior_rc, e.held_resources)
post_cp ← bayesian_update(costModel[p.type].prior_cp,
(e.priority, e.in_cycle_degree))
cost ← w_rt·E[post_rt] + w_rc·E[post_rc] + w_cp·E[post_cp]
costHeap.insert((cost, p.pid))

// Step 3: Extract victim
(_, victim_pid) ← costHeap.extractMin()
preempt_process(victim_pid)
return victim_pid
```

After preemption, the system logs the outcome (actual restart cost incurred) and performs an online update of the relevant prior distribution parameters, enabling continuous model refinement [12–14].

COMPUTATIONAL COMPLEXITY ANALYSIS

Table 2 compares the asymptotic complexity of the IGP with that of the baseline heuristics across the major algorithmic phases. The notation uses V = vertices in the dependency graph, E = edges, n = processes in the deadlock cycle, and k = number of distribution parameters per factor.

The dominant cost in IGP beyond static heuristics is the O(n log n) min-heap sort over the expected costs. For typical deadlock cycle sizes ($n \leq 8$) observed in production multiprocessor workloads [15, 16], this overhead becomes negligible. The space overhead O(V + k) accounts for the graph structure and the distribution parameter store; k is a small constant (two parameters per distribution, three distributions per-process class) and does not scale with the system size.

The online update step O(k·n) represents the per-event parameter refresh and similarly scales only with the cycle size, not with the total system process count. Conjugate prior families guarantee that this update requires no numerical integration, making the approach suitable for real-time and embedded multiprocessor environments, where scheduling latency budgets are tight [7, 17].

EVALUATION

Simulation Setup

The evaluation employs SimPy, a Python-based discrete-event simulation framework, to model the process execution, resource allocation, and deadlock detection in a controlled multiprocessor environment. The simulator implements both the RAG adjacency list and the WFG DFS cycle detector described in Section 2.

Three process classes were simulated: I/O-bound (short CPU bursts, frequent waits), CPU-bound (long computational bursts), and transactional (moderate computation and elevated criticality). The simulation spanned 10,000-time units with process counts ranging from 50 to 200 across 5–10 resource types at moderate deadlock frequency, repeated over 10 random seeds for statistical reliability.

Table 2. Algorithmic complexity comparison across deadlock resolution strategies.

Algorithm	Deadlock detection	Victim selection	Update/learning	Space
Youngest/oldest	O(V + E)	O(n)	None	O(V)
Lowest priority	O(V + E)	O(n)	None	O(V)
IGP–Bayesian (proposed)	O(V + E)	O(n log n)	O(k·n)	O(V + k)

The IGP is benchmarked against three static heuristics: Youngest (preempt the most recently created process), Oldest (preempt the longest-running process), and lowest priority (preempt the process with the lowest scheduling priority) [4, 18]. Performance is measured using three key performance indicators (KPIs):

- *Average resolution overhead:* Total CPU time wasted across all deadlock preemptions.
- *Victim fairness (Std Dev):* Standard deviation of per-process preemption counts; lower values indicate balanced selection.
- *Adaptability:* Time units required to stabilize overhead after a midpoint workload shift (70% I/O-bound → 70% CPU-bound).

Results

Table 3 presents the aggregate performances across all simulation runs. Figures 1–4 illustrate the individual performance dimensions.

IGP achieves the lowest average resolution overhead (95 CPU time units) compared to 145 (youngest), 135 (lowest priority), and 120 (oldest), representing a 34% improvement over the best static heuristic (lowest priority). The min-heap selection mechanism, guided by Bayesian posterior estimates, consistently avoids processes with high remaining execution times or critical dependencies, which are the primary contributors to costly preemptions [19].

On victim fairness, the IGP records a standard deviation of 2.1, marginally better than the lowest priority (2.2) and substantially better than the youngest (2.9). The probabilistic cost model avoids systematic bias toward any single process class because evidence-driven selection naturally distributes preemptions across processes, the cost of which is situationally the lowest [6].

The most pronounced advantage of IGP is its adaptability. When the workload composition shifts at the simulation midpoint, the Bayesian priors are updated based on the new evidence stream. The IGP stabilizes in approximately 850-time units compared to 1,900–2,200 for static heuristics, which have no learning mechanism. The min-heap reorders victim candidates immediately as the updated posteriors change the expected costs, without requiring a restart or reconfiguration [12].

Table 3. Aggregate performance comparison across victim selection algorithms.

Algorithm	Avg. resolution overhead	Fairness (std. dev.)	Post-shift stabilization (time units)
Youngest	145	2.9	~2,200
Oldest	120	2.5	~1,900
Lowest priority	135	2.2	~2,100
IGP (proposed)	95	2.1	~850

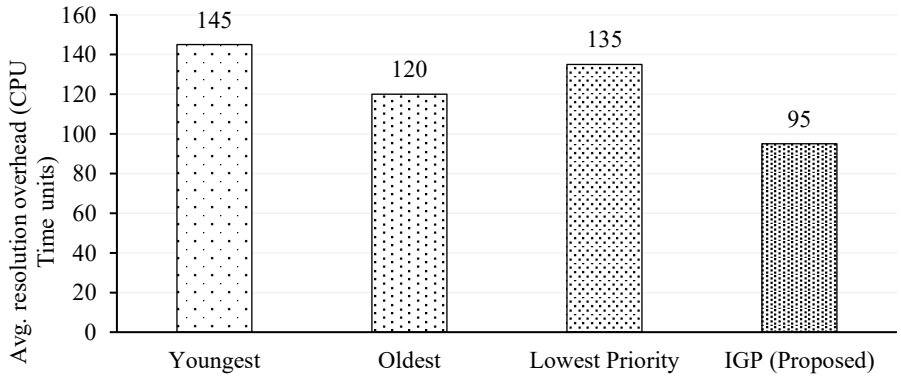

Figure 1. Average resolution overhead comparison. IGP achieves 95 CPU time units, a 34% reduction over the best static heuristic (oldest, 120 units).

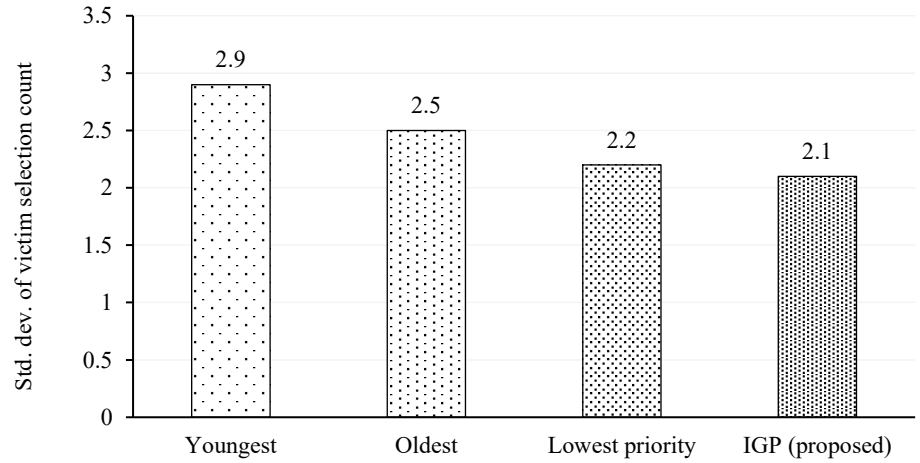

Figure 2. Victim fairness is measured by the standard deviation of preemption counts across processes. Lower values indicate more balanced victim selection.

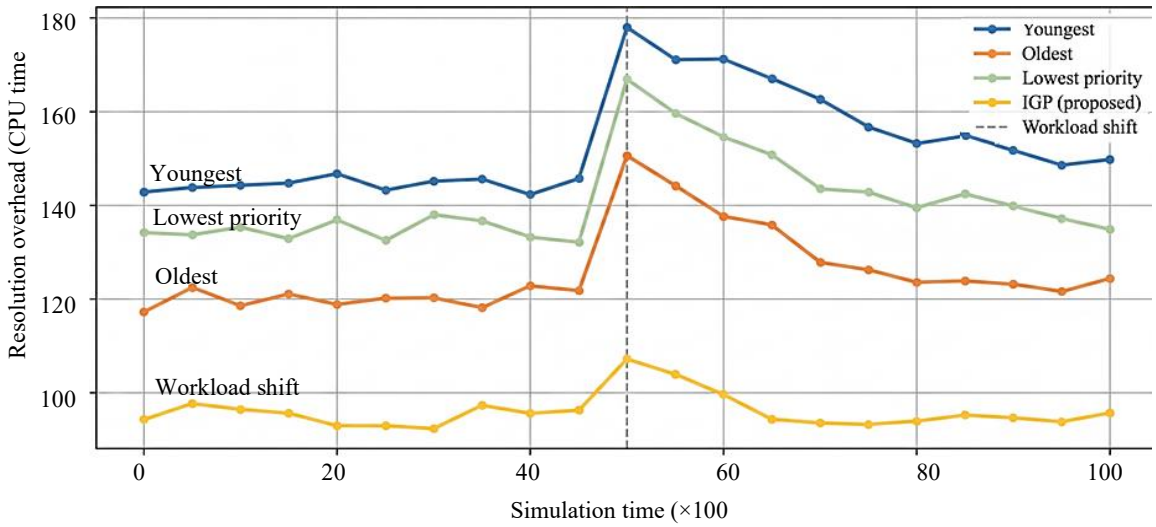

Figure 3. Resolution overhead over simulation time. The workload shifts at t = 50×100 = 5,000 units. IGP recovers to baseline overhead ~2.5× faster than the best static heuristic (oldest).

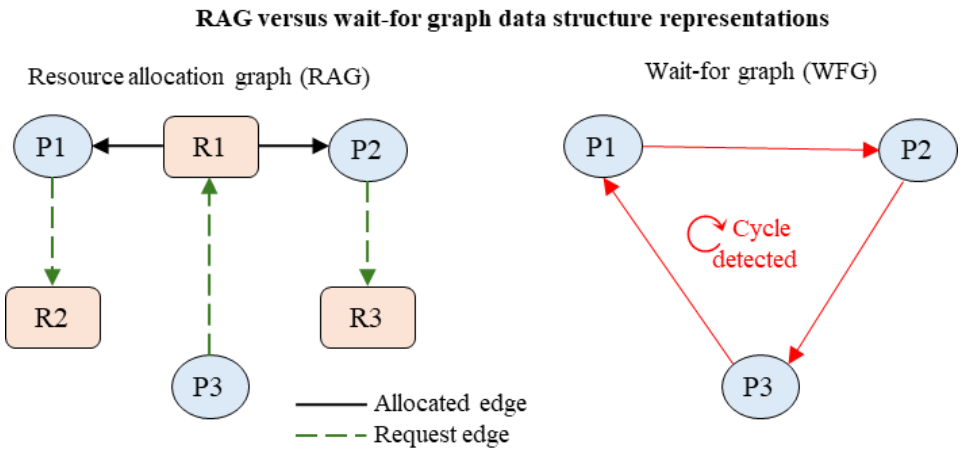

Figure 4. *Left*—Resource allocation graph (RAG) with bipartite structure (processes as circles, resources as rectangles). *Right*—Wait-for graph (WFG) projected from RAG, revealing the circular wait cycle directly.

Figure 4 illustrates the structural distinction between the RAG and WFG, the two core graph data structures underlying deadlock detection. The RAG includes resource nodes and distinguishes allocation from request edges, whereas the WFG compresses resource nodes into direct process-to-process wait relationships, enabling faster DFS cycle detection at the cost of losing resource-level granularity used by the cost model.

Taken together, the results confirm all three evaluation hypotheses: IGP reduces resolution overhead (H1), maintains fairness comparable to or better than the baselines (H2), and adapts significantly faster to workload shifts (H3). The structural choices, such as adjacency list graphs, hash map evidence stores, conjugate distribution objects, and min-heap victim extraction, collectively enable these improvements within a tractable computational budget [20–22].

CONCLUSION AND FUTURE WORK

This study presents a data-structure-centric analysis of IGP, a probabilistic deadlock resolution algorithm for multiprocessor systems. By mapping the IGP lifecycle onto a coherent pipeline of directed graphs (RAG, WFG), adjacency lists, hash maps, Gaussian distribution objects, and min-heaps, we have shown that Bayesian deadlock resolution is not merely a theoretical framework but an architecturally sound approach that can be implemented with standard operating system data structure primitives.

The key structural innovations are as follows: (1) the use of adjacency list RAG and WFG for O(V + E) deadlock detection; (2) hash map evidence stores for O(1) per-process data retrieval; (3) conjugate Gaussian distribution objects for O(1) Bayesian updates; and (4) a min-heap for O(n log n) optimal victim extraction. The simulation results validate a 34% reduction in resolution overhead, maintained fairness, and 2.5× faster workload shift adaptation compared to the best static heuristic.

Several directions remain for future research. First, the integration of the adjacency list WFG and IGP min-heap into a Linux kernel module enables evaluation under real workloads and interrupt constraints. Second, replacing Gaussian conjugate priors with non-parametric density estimators (e.g., kernel density estimation) would allow the model to capture multimodal cost distributions arising from heterogeneous workload mixes. Third, the RAG structure can be extended to model proactive deadlock avoidance by predicting cycle formation before it is completed using graph-theoretic cycle prediction algorithms combined with Bayesian risk estimation. Fourth, applying this framework to distributed multiprocessor systems, where the WFG is partitioned across nodes, introduces a communication-efficient distributed cycle detection as a natural extension.

Overall, the fusion of classical graph-based deadlock modelling with probabilistic Bayesian inference, implemented using efficient data structures, provides a robust architectural foundation for next-generation adaptive resource management in concurrent computing systems.

REFERENCES

1. Zhang J, Chen Z, Ye Y, Chen H, Han X, Jiang J, et al. IPDR: An inter-chiplet priority-driven deadlock resolution for 2-D/2.5-D multichiplet systems. IEEE Trans Very Large Scale Integr VLSI Syst. 2025;33:2424–2437. doi:10.1109/TVLSI.2025.3583289.
2. Müller M. Multi-agent reinforcement learning for deadlock handling among autonomous mobile robots [Preprint]. 2025. arXiv:2511.07071. doi:10.48550/arXiv.2511.07071
3. Chen Y, Wang C, Guo M, Li Z. Multi-robot trajectory planning with feasibility guarantee and deadlock resolution: An obstacle-dense environment. IEEE Robot Autom Lett. 2023;8(4):2197–2204. doi:10.1109/LRA.2023.3248377.
4. Yang Y, Li T, Dai Y, Wang B, Ma S, Sun Y. Absorb: Deadlock resolution for 2.5D modular chiplet based systems. In: Tari Z, Li K, Wu H, editors. Algorithms and Architectures for Parallel Processing: 23rd International Conference, ICA3PP 2023, Tianjin, China. 2024. p. 474–487. doi:10.1007/978-981-97-0834-5_27.

5. GeeksforGeeks. (2017). Resource allocation graph (RAG) [Online]. GeeksforGeeks. Available from: https://www.geeksforgeeks.org/operating-systems/resource-allocation-graph-rag-in operat ing-system/.
6. Le Ngoc H, Cong HT. Enhancing load balancing in cloud computing through deadlock prediction. In: Vo NS, Tran HA, editors. Industrial Networks and Intelligent Systems. INISCOM 2023. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering. Vol. 531. Cham: Springer Nature Switzerland; 2023. p. 257–274. doi:10.1007/978-3-031-47359-3_19.
7. Wijnings PWA, Roa-Villescas M, Stuijk S, de Vries B, Corporaal H. On the importance of the execution schedule for Bayesian inference. ACM Trans Probab Mach Learn. 2024;1:1–28. doi:10.1145/3690830.
8. Tang S, Wu T, Tian Y, Wang H, Wu Y, Jiang R, et al. A Bayesian inference-enhanced evolutionary algorithm for sleep scheduling of software-defined radio sensors. In: Huang DS, Chen W, Pan Y, Chen H, editors. Advanced Intelligent Computing Technology and Applications. ICIC 2025. Lecture Notes in Computer Science. Vol. 15851. Singapore: Springer; 2025. p. 1–14. doi:10.1007/978-981-96-9849-3_1.
9. Peng B, Xie Y, Seco-Granados G, Wymeersch H, Jorswieck EA. Communication scheduling by deep reinforcement learning for remote traffic state estimation with Bayesian inference. IEEE Trans Veh Technol. 2022;71(4):4287–4300. doi:10.1109/TVT.2022.3145105.
10. Kim DY, Kim RG, Kwak HS. A closed-loop scheduling framework for prefabricated bridge girders: Bayesian regression and TCTO-based optimization. Buildings. 2025;15:4168. doi:10.3390/buildings15224168.
11. Nguyen Trong T, Cuong NHV, Pham TV, Cuong NHH, Khiet BT. An approach to new technical solutions in resource allocation based on artificial intelligence. In: Mohanty SN, Garcia Diaz V, Satish Kumar GAE, editors. Intelligent Systems and Machine Learning: First EAI International Conference, ICISML 2022, Hyderabad, India. Switzerland: Springer Nature. 2023. p. 325–334. doi:10.1007/978-3-031-35081-8_27.
12. Mandal SK, Ogras UY, Doppa JR, Ayoub RZ, Kishinevsky M, Pande PP. Online adaptive learning for runtime resource management of heterogeneous SoCs. 57th ACM/IEEE Design Automation Conference (DAC), San Francisco, California, USA. 2020. p. 1–6. doi:10.1109/DAC18072.2020.9218604.
13. Lee JJ, Mooney VJ III. Hardware/software partitioning of operating systems: focus on deadlock detection and avoidance. IEE Proc Comput Digit Tech. 2005;152(2):167–182. doi:10.1049/ip-cdt:20045078.
14. Shankaran N, Kinnebrew JS, Koutsoukas XD, Lu C, Schmidt DC, Biswas G. An integrated planning and adaptive resource management architecture for distributed real-time embedded systems. IEEE Trans Comput. 2009;58:1485–1499. doi:10.1109/TC.2009.44.
15. Helmy T. An improved deadlock detection and resolution algorithm for distributed computing systems. Preprints. 2024. doi:10.20944/preprints202403.1310.v1.
16. Shanu S, Sastry HG, Marriboyina V. Optimal solution approach on large scale data to avoid deadlocks in resource allocations. Mater Today Proc. 2021;47:7162–7166. doi:10.1016/j.matpr.2021.06.357.
17. Luo XG, Zhou L, Wang R. Manufacturing process based on recognition rules and Bayesian networks. Int J Simul Model. 2025;24(1):135–146. doi:10.2507/IJSIMM24-1-CO2.
18. Feng Y, Ren S, Cao Y, Xing K, Yang Y. Deadlock control for flexible assembly systems with multiple resource requirements and separately-loaded parts. IEEE Trans Autom Sci Eng. 2025;22:9275–9284. doi:10.1109/TASE.2024.3504714.
19. Rogalska M, Hejducki Z, Kostrzewa-Demczuk P. Causal reasoning in construction process scheduling. Appl Sci. 2025;16(1):207. doi:10.3390/app16010207.
20. Venkatesh S, Smith JS. A graph-theoretic, linear-time scheme to detect and resolve deadlocks in flexible manufacturing cells. J Manuf Syst. 2003;22(3):220–238. doi:10.1016/S0278-6125(03)90022-7.

21. Gabriel PHR, Albertini MK, Castelo A, De Mello RF. Min-heap-based scheduling algorithm: an approximation algorithm for homogeneous and heterogeneous distributed systems. Int J Parallel Emergent Distrib Syst. 2016;31(1):64–84. doi:10.1080/17445760.2015.1009067.
22. Naithani A, Eyerman S, Eeckhout L. Reliability-aware scheduling on heterogeneous multicore processors. 2017 IEEE International Symposium on High Performance Computer Architecture (HPCA), Austin, TX, USA. 2017. p. 397–408. doi:10.1109/HPCA.2017.12.

