



# Journal of Applied Science

Biannual Peer Reviewed Journal Issued by Research  
and Consultation Center , Sabratha University

Issue (12)  
April 2024





# Journal of Applied Science

Biannual Peer Reviewed Journal Issued by Research and Consultation Center,  
Sabratha University

## Editor

Dr. Hassan M. Abdalla

## Associate Editors

Dr. Mabruk M. Abogderah

Dr. Jbireal M. Jbireal

Dr. Ali K. Muftah

Dr. Esam A. Elhefian

## English Language Reviewer

Dr. Siham S. Abdelrahman

## Arabic Language Reviewer

Dr. Ebrahim K. Altwade

## Designed By

Anesa M. Al-najeh

## **Editorial**

We start this pioneering work, which do not seek perfection as much as aiming to provide a scientific window that opens a wide area for all the distinctive pens, both in the University of Sabratha or in other universities and research centers. This emerging scientific journal seeks to be a strong link to publish and disseminate the contributions of researchers and specialists in the fields of applied science from the results of their scientific research, to find their way to every interested reader, to share ideas, and to refine the hidden scientific talent, which is rich in educational institutions. No wonder that science is found only to be disseminated, to be heard, to be understood clearly in every time and place, and to extend the benefits of its applications to all, which is the main role of the University and its scholars and specialists. In this regard, the idea of issuing this scientific journal was the publication of the results of scientific research in the fields of applied science from medicine, engineering and basic sciences, and to be another building block of Sabratha University, which is distinguished among its peers from the old universities.

As the first issue of this journal, which is marked by the Journal of Applied Science, the editorial board considered it to be distinguished in content, format, text and appearance, in a manner worthy of all the level of its distinguished authors and readers.

In conclusion, we would like to thank all those who contributed to bring out this effort to the public. Those who lit a candle in the way of science which is paved by humans since the dawn of creation with their ambitions, sacrifices and struggle in order to reach the truth transmitted by God in the universe. Hence, no other means for the humankind to reach any goals except through research, inquiry, reasoning and comparison.

### **Editorial Committee**

## Notice

The articles published in this journal reflect the opinions of their authors only. They are solely bearing the legal and moral responsibility for their ideas and opinions. The journal is not held responsible for any of that.

Publications are arranged according to technical considerations, which do not reflect the value of such articles or the level of their authors.


### Journal Address:


Center for Research and Consultations, Sabratha University

Website: <https://jas.sabu.edu.ly/index.php/asjsu>

Email: [jas@sabu.edu.ly](mailto:jas@sabu.edu.ly)

**Local Registration No. (435/2018)**

ISSN  2708-7301

ISSN  2708-7298

## Publication instructions

The journal publishes high quality original researches in the fields of Pure Science, Engineering and Medicine. The papers can be submitted in English or Arabic language through the Journal email ([jas@sabu.edu.ly](mailto:jas@sabu.edu.ly)) or CD. The article field should be specified and should not exceed 15 pages in single column.

All submitted research manuscripts must follow the following pattern:

- Title, max. 120 characters.
- Author Name, Affiliation and Email
- Abstract, max. 200 words.
- Keywords, max. 5 words.
- Introduction.
- Methodology.
- Results and Discussion.
- Conclusion.
- Acknowledgments (optional).
- References.

### Writing Instructions:

Papers are to be submitted in A4 (200×285 mm) with margins of 25 mm all sides except the left side, which should be 30 mm. Line spacing, should also be 1.15.

**Table 1. Font size and style**

	<b>Bold</b>	<b>English</b>	<b>Arabic</b>
Font Style	✓	Times New Roman	Simplified Arabic
Article Title	✓	14 Capital	16
Authors Name	✓	12	14
Affiliation	×	11	13
Titles	✓	12	14
Sub-Title	✓	12	13
Text	×	12	14
Figure Title	✓	11	13
Table Title	✓	11	13
Equations	✓	12	14

### Figures:

All figures should be compatible with Microsoft Word with serial numerals. Leave a space between figures or tables and text.

### References:

The references should be cited as Harvard method, eg. Smith, R. (2006). References should be listed as follows:

**Articles:** Author(s) name, Year, Article Title, Journal Name, Volume and Pages.

**Books:** Author(s) name. Year. "Book title" Location: publishing company, pp.

**Conference Proceedings Articles:** Author(s) name. Year. "Article title". Conference proceedings. pp.

**Theses:** Author(s) name. Year. "Title". Degree level, School and Location.

**Invitation**

The Editorial Committee invites all researchers "Lectures, Students, Engineers at Industrial Fields" to submit their research work to be published in the Journal. The main fields targeted by the Journal are:

- Basic Science.
- Medical Science & Technology.
- Engineering.

**Refereeing**

The Editorial Committee delivers researches to two specialized referees, in case of different opinions of arbitrators the research will be delivered to a third referee.

**Editorial Committee**

Dr. Hassan M. Abdalla.  
Dr. Mabruk M. Abogderah.  
Dr. Jbireal M. Jbireal.  
Dr. Ali K. Muftah.  
Dr. Esam A. Elhefian.  
Dr. Siham S. Abdelrahman.  
Dr. Ebrahim K. Altwade.  
Anesa M. Al-najeh.

## CONTENTS

[1]	STUDY OF THE ACTIVE INGREDIENT OF FOUR DIFFERENT BRANDS OF COMMERCIAL DICLOFENAC SODIUM OF SELECTED PHARMACIES IN THE WESTERN REGION OF LIBYA.....	1
[2]	COMPARING SOLVING LINEAR PROGRAMMING PROBLEMS WITH APPLICATIONS OF THE MOORE-PENROSE GENERALIZED INVERSE TO LINEAR SYSTEMS OF ALGEBRAIC EQUATIONS .....	9
[3]	NITRATES IN MAN-MADE RIVER COMPARED TO GROUNDWATER WELLS IN EL-KUFRA AREA.....	19
[4]	DEGRADATION OF REACTIVE BLACK 5 DYE IN WATER FALLING FILM DIELECTRIC BARRIER DISCHARGE REACTOR (DBD).....	32
[5]	ANTIMICROBIAL RESISTANCE IN UROPATHOGEN ISOLATES FROM PATIENTS WITH URINARY TRACT INFECTIONS .....	44
[6]	SMOOTHING EFFECTS FOR A MODEL OF QUASI GEOSTROPHIC EQUATION.....	53
[7]	CLASSIFICATION OF BREAST CANCER USING MACHINE LEARNING ALGORITHMS.....	60
[8]	ENERGY-EFFICIENT INTRUSION DETECTION IN WSN: LEVERAGING IK-ECC AND SA-BILSTM .....	72
[9]	THE EFFECT OF ADDING POLYMER COMPOUNDS TO METALS ON ITS MECHANICAL PROPERTIES.....	89
[10]	EXPLORING DEADLOCK DETECTION ALGORITHMS IN CONCURRENT PROGRAMMING: A COMPARATIVE ANALYSIS AND EVALUATION .....	106
[11]	ISOLATION AND DETECTION OF BACTERIAL SPECIES CAUSING GINGIVITIS IN PATIENTS WITH TYPE 2 DIABETES.....	118

## **EXPLORING DEADLOCK DETECTION ALGORITHMS IN CONCURRENT PROGRAMMING: A COMPARATIVE ANALYSIS AND EVALUATION**

**Ahmed Elajeli Rgibi<sup>1\*</sup>, Abdusamea Ibrahim Omer<sup>2</sup>, Amany Khalifa Alarbish<sup>3</sup>,  
and Amal Apojila Osha<sup>4</sup>**

<sup>1,2,3,4</sup> Department of computer Engineering, Faculty of Engineering, Sabratha University,  
Libya

\* [ahmed.rgibi@sabu.edu.ly](mailto:ahmed.rgibi@sabu.edu.ly)

### **Abstract**

Concurrency in computing systems brings numerous benefits, but it also introduces challenges, including the potential occurrence of deadlocks. Deadlocks arise when multiple processes are unable to proceed due to each waiting for a resource held by another process, halting system progress and leading to resource wastage. The knowledge gap in deadlock detection in concurrent programming involves understanding various techniques and algorithms to identify and prevent deadlocks. Also, distinguish the difference between deadlock detection algorithms. In this work, the deadlock detection process and various deadlock detection algorithms used in concurrent programming were explored. Additionally, a variety of these algorithms were discussed in depth, including the banker's algorithm, the wait-for graph algorithm, and the resource allocation graph algorithm. It was also provided with a comprehensive explanation to help understand how it works. Moreover, an analysis was conducted on the strengths, weaknesses, and performance characteristics of different deadlock detection techniques.

**Keywords:** deadlock; algorithms; detection; RAG; FWG; graph.

### **Introduction**

Concurrency is a fundamental characteristic of modern computing systems, where multiple tasks or processes execute concurrently, aiming to enhance system performance and resource utilization. However, concurrency can introduce various challenges, including the possibility of deadlocks. Deadlocks occur when multiple processes are unable to proceed because each is waiting for a resource held by another process within the system. This situation can result in system-wide halts and resource wastage. To cope with deadlocks, it is crucial to employ effective deadlock detection algorithms that can identify and handle deadlock situations. Deadlock detection algorithms are designed to periodically examine the system's resource allocation graph or resource hierarchy to determine the presence of a deadlock. Once a deadlock is



detected, appropriate actions can be taken to resolve the deadlock and allow the system to continue functioning. Given the importance of efficient and accurate deadlock detection algorithms, deadlock can occur if and only if all four of these conditions are fulfilled: (1) **Mutual Exclusion**: At least one resource in the system must be held in a non-shareable mode, where only one process can access the resource at a time. This condition ensures that once a process has acquired a resource, no other process can access it until it is released Rgibi et al. (2022). (2) **Hold and Wait**: A process must be holding at least one resource while waiting to acquire additional resources. This condition creates a situation where processes can wait indefinitely for resources to be released while holding resources themselves. (3) **Circular Wait**: A circular chain of two or more processes exists, where each process is waiting for a resource held by another process in the chain. This forms a circular dependency, creating a deadlock situation Acosta et al. (2013).

## Related Work

Reviewing related work is an essential part of any research paper, as it demonstrates the existing knowledge and research efforts in the given field. Here are some possible related works for a paper reviewing deadlock detection algorithms in concurrent programming:

- Nabeel et al. (2023): "Comparative Analysis of Deadlock Detection Algorithm based on Blockchain" by This paper presents a comparative analysis of various deadlock detection algorithms in blockchain distributed systems. It explores the strengths and weaknesses of different techniques and evaluates their performance in detecting and resolving deadlocks.
- kate et al. (2016): "A survey on distributed deadlock and distributed algorithms to detect and resolve deadlock". This survey paper provides an overview of deadlock detection and recovery algorithms, specifically in the context of operating systems. It discusses different approaches, including resource allocation graphs, wait-for graphs, and cycle detection algorithms, along with their pros and cons.
- Nasseri et al. (2018): "Concurrency control methods in distributed database: A review and comparison". Although this paper focuses on concurrency control in distributed databases, it also covers deadlock detection mechanisms. It discusses distributed deadlock detection algorithms and their effectiveness in resolving deadlocks in a distributed computing environment.
- R. Agarwal et al. (2010): "Detection of deadlock potentials in multithreaded programs". This research paper presents a comprehensive analysis and comparison of various deadlock detection algorithms in multithreaded

programs. It assesses factors such as accuracy, scalability, and overheads, offering insights into the performance of different algorithms.

- Haque et al. (2017): "A Survey on Deadlock Detection and Prevention Techniques in Real-Time Systems". Focusing on real-time systems, this survey paper covers deadlock detection and prevention techniques. It reviews different approaches, such as resource allocation graphs, control flow analysis, and scheduling algorithms, to address deadlocks in time-critical systems.

These related works provide a foundation for understanding deadlock detection algorithms in concurrent programming. By reviewing and comparing their approaches, limitations, and performance, a comprehensive understanding of the existing research landscape can be obtained, thereby facilitating the identification of research gaps and the formulation of novel contributions in the paper at hand.

### How to Avoid Deadlock

There are four techniques used to avoid deadlocks. These techniques are:

**Deadlock prevention:** aims at eliminating one or more of the necessary conditions for deadlocks.

**Deadlock avoidance:** uses resource allocation strategies to ensure that the system does not enter an unsafe state.

**Deadlock detection:** involves periodically checking the resource allocation graph or using algorithms to identify cycles and take appropriate measures to resolve the deadlock.

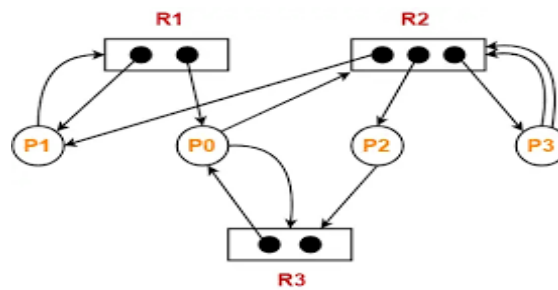
**And don't do anything.**

### Dead Detection Process

Deadlock detection is an approach in concurrent programming used to identify the presence of a deadlock in a system. It involves periodically checking the resource allocation graph or using sophisticated algorithms to analyze the system state and identify any cycles that indicate a deadlock situation. Rgibi et al. (2022). Here is an overview of the deadlock detection process in concurrent programming:

### Resource Allocation Graph (RAG)

Deadlock detection starts with representing the system's resources and processes as a resource allocation graph. Each process is a node, and each resource is represented as a resource type with instances or units, as shown in Figure (1).



**Figure (1): Resource Allocation Graph.**

### Graph Construction

The resource allocation graph is constructed by adding edges between processes and resources based on their allocation and request relationships. The edges indicate which processes hold or request which resources.

### Cycle Detection

The DFS algorithm checks if there is a cycle in the WFG. If a cycle is present, it indicates the possibility of a deadlock. One popular algorithm used for cycle detection in directed graphs is the depth-first search (DFS) algorithm. By performing DFS on the WFG, it searches for back edges that form a cycle. Here are the DFS algorithm steps:

```
DFS (G, U)
    u.visited = true
    for each V ? G.ADJ[U]
        if v.visited == false
            DFS(G, V)
init() {
    for each U ? G
        u.visited = false
    for each u ? g
        DFS(G, U)
```

And the following code checks if there is a Cycle detection in WFG.

```
bool isCyclic() {
    for (int i = 0; i < vertices; i++) {
        visited[i] = false;
        recursionStack[i] = false;
    }
    for (int i = 0; i < vertices; i++) {
        if (hasCycle(i))
```

```
return true;  }  
return false; }
```

### Algorithm Execution

Various algorithms can be used to detect cycles in the resource allocation graph. These algorithms can be broadly categorized as graph-based algorithms or state-based algorithms. Here is an overview of the deadlock category:

**Graph-based algorithms:** such as the cycle detection algorithm based on depth-first search (DFS) or the Banker's algorithm, explore the resource allocation graph to identify cycles or unsafe states.

**State-based algorithms:** employ techniques like model checking, formal verification, or state-space exploration to analyze the state of the system and identify potential deadlocks.

### Deadlock Resolution

Once a deadlock is detected, appropriate actions can be taken to resolve it. This may involve aborting one or more processes in the deadlock cycle, rolling back their operations, or requesting additional resources from external sources.

### Deadlock Detection Algorithms

Deadlock detection algorithms are used in computer systems to identify and manage deadlock situations, where processes are unable to progress because they are waiting for resources held by other processes. These algorithms aim at detecting the presence of deadlocks in a system and take appropriate actions to resolve them. In the next section, we will explain the most popular algorithms used to detect deadlock:

**Cycle Detection in the WFG:** Cycle detection algorithm checks if there is a cycle in the WFG. If a cycle is present, it indicates the possibility of a deadlock. One popular algorithm used for cycle detection in directed graphs is the depth-first search (DFS) and breadth-first search (BFS) algorithm. By performing DFS on the WFG, it searches for back edges that form a cycle. Figure (2) shows whether there is a cycle or not in RAG.

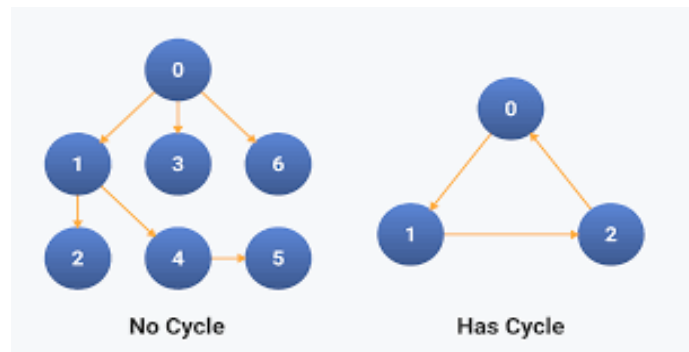


Figure (2): Resource Allocation Graph.

**Reduction and Simplification:** This technique simplifies the WFG by reducing it to a smaller and more manageable graph. The idea is to remove unnecessary edges or nodes that do not contribute to the presence of a deadlock. This simplification helps improve the efficiency of deadlock detection algorithms. The following steps explain how to reduce WFG to a smaller and more manageable graph:

- **Identify and remove any unnecessary edges:** Review the WFG and identify any dependencies that are not required. These may include redundant or unnecessary dependencies between processes and tasks. Remove these edges to simplify the graph. Identifying dependencies that are not required in a Wait-For-Graph (WFG) can be done through careful analysis and understanding of the system or project (Hassan et al., 2022).
- **Merge similar or equivalent processes:** Identify processes that have similar functionality or are equivalent in terms of dependencies and merge them into a single process. This helps to reduce the number of vertices in the WFG and simplify the graph structure.
- **Identify and remove any transitive dependencies:** Look for transitive dependencies in the WFG, which occur when there is a chain of dependencies between processes  $A \rightarrow B \rightarrow C$ . If process A is dependent on process C, the dependency between A and B becomes redundant. Remove these transitive dependencies to simplify the WFG.
- **Apply reduction rules:** Different reduction rules can be applied to simplify the WFG further. Some common reduction rules include: remove self-loops, remove redundant edges, and collapse parallel edges.

**Optimization Algorithm:** Various optimization techniques can be applied to improve the efficiency of deadlock detection algorithms. This includes using optimization algorithms like Tarjan's strongly connected components algorithm, which identifies

strongly connected components in the WFG and helps in reducing the graph size and analysis complexity.

**Incremental Deadlock Detection Algorithm:** In large-scale systems, continuously analyzing the entire WFG for deadlocks can be computationally expensive. Incremental deadlock detection techniques aim to reduce the computational overhead by focusing on changes in the WFG. By tracking only the relevant changes, such as newly added edges or updated resource allocations, the deadlock detection algorithm can avoid analyzing the entire WFG repeatedly. The Incremental Deadlock Detection algorithm is used to detect deadlocks in a system. The algorithm works as follows:

1. Initialize data structures.
2. Check for a new resource allocation or release.
3. Check for cycles in the lock graph.
4. Check for a cycle in the wait-for graph.
5. Resolve and handle deadlocks.
6. Repeat steps 2-5.
7. Termination.

It's important to note that the implementation details of the Incremental Deadlock Detection algorithm can vary depending on the specific system and programming language being used.

**Wait-Die and Wound-Wait Algorithm:** Wait-Die and Wound-Wait are two deadlock detection algorithms commonly used in resource allocation with a dynamic allocation policy. These algorithms employ a wait-for graph and a timestamp mechanism to detect deadlocks. Wait-Die allows older processes to wait for younger processes, while Wound-Wait allows younger processes to kill older processes if a deadlock is detected. As shown in Figure (3), these algorithms provide a dynamic and efficient solution to deadlock detection, particularly in scenarios where resource allocation changes frequently.

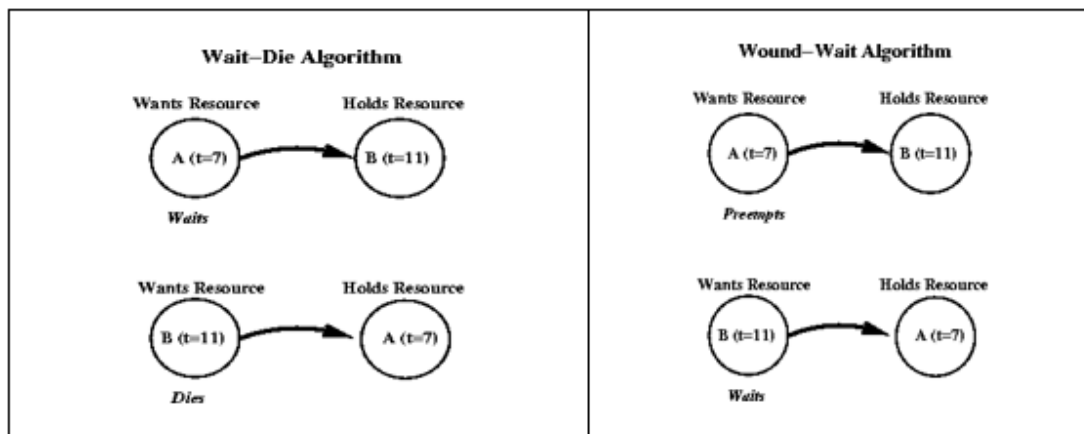


Figure (3): Wound-Wait and Wait-Die Algorithms.

- The Wait-Die algorithm pseudo code can be described as follows:
  1. Input: A requesting process P with timestamp  $T_i$
  2. If there exists a process Q with a timestamp  $T_j$  such that  $T_j < T_i$ :
  3. Let process P wait.
  4. Else:
  5. Grant the resource to process P.
  6. End Procedure
- The Wound-Wait algorithm pseudo code can be described as follows:
  1. Input: A requesting process P with timestamp  $T_i$
  2. If there exists a process Q with timestamp  $T_j$  such that  $T_j < T_i$ :
  3. Let process Q die.
  4. Grant the resource to process P.
  5. Else:
  6. Let process P wait.
  7. End Procedure

**Banker's Algorithm:** The Banker's algorithm is another widely used deadlock detection algorithm. It takes a different approach by considering the available resources and the maximum resource demands of each process. The algorithm

simulates resource allocation and checks if any process can safely complete its execution based on the available resources. If no safe sequence is found, a deadlock is confirmed (Zarnay et al., 2014). The banker's algorithm ensures resource allocation is done in a way that prevents deadlocks from occurring. Here are the steps involved in this algorithm:

1. **Initialization:** The algorithm begins by initializing the available resources, the maximum resources per process, and the current allocation of resources to each process.
2. **Request Handling:** When a process requests a resource, the system checks if granting the request will result in a safe state.
3. **Safety Algorithm:** The banker's algorithm uses a safety algorithm to determine if the system can safely grant the resource request. It starts by assuming all processes are in a safe state and creating a work queue to track the available resources.
4. **Resource Release:** When a process completes its execution, release the allocated resources by updating the allocation matrix and increasing the available resources accordingly.

These steps are repeated every time a process requests or releases resources to ensure that the system remains in a safe state. By carefully managing resource allocation, the Banker's algorithm aims to prevent deadlock situations in concurrent systems.

### Identify and evaluate dependencies in a WFG

To evaluate dependencies in a WFG, the following approach can be considered:

**Understand the requirements:** Have a clear understanding of the requirements of the system. This will help you identify dependencies that are essential for the correct functioning of the system and those that are not required.

**Review the business or functional logic:** Examine the business or functional logic of the processes/tasks in the WFG. Identify dependencies that are unnecessary based on the logic and requirements. Look for processes/tasks that can function independently or don't require certain dependencies to execute successfully.

**Evaluate data flow:** Analyze the data flow between processes in the WFG. Identify dependencies where the flow of data is not necessary or can be optimized. Determine if there are any redundant dependencies that don't contribute to the desired output or outcome.

**Consider time and performance constraints:** Assess the time and performance constraints of the project or system. Identify dependencies that can be removed without



adversely affecting the overall performance or time constraints. This could include processes that can be executed in parallel or dependencies that can be modified to optimize performance.

**Discuss with stakeholders:** Consult with stakeholders, including project managers, developers, and domain experts. Discuss the dependencies in the WFG and seek their input on identifying dependencies that are not required. Their insights and expertise can provide valuable guidance in evaluating and eliminating unnecessary dependencies.

### **Illustrative Example**

To illustrate deadlock detection in concurrent processes: Let's consider a scenario where two processes, Process A and Process B, are running concurrently and share two resources: Resource 1 and Resource 2.

1. Process A starts and requests Resource 1, which is available.
2. Then, Process A tries to request Resource 2, but it is currently being used by Process B. So, Process A has to wait for Process B to release Resource 2.

Two concurrent processes, Process A and Process B, share resources in the scenario described. Process A requests Resource 1, which is available, but Resource 2, needed by Process A, is currently being used by Process B. Similarly, Process B requests Resource 2, which is not yet available, while Process A has Resource 1. This situation creates a deadlock, as both processes are unable to proceed. Deadlock detection algorithms are used to monitor resource requests and identify circular wait conditions. In this case, the algorithm would identify the circular wait between Process A and Process B and take appropriate actions such as terminating a process, reallocating resources, or reordering requests to prevent the deadlock. Implementing these algorithms proactively resolves potential deadlocks, enhancing system reliability and performance.

### **Results**

In concurrent programming, sharing resources among processes can lead to the occurrence of deadlocks. Even nevertheless, deadlock detection algorithms have improved over time. The Cycle Detection in the WFG, Reduction and Simplification, Algorithm Optimisation, and Incremental Deadlock Detection Algorithm given in greater depth are in line with our definition of correctness as compared to classical algorithms and when defining correctness.

## Conclusion

In conclusion, the review of deadlock detection algorithms for concurrent programming highlights the importance and complexity of managing deadlocks in computer systems. The variety of algorithms available, such as the banker's algorithm, wait-for graph algorithm, resource allocation graph algorithm, and deadlock detection using resource vectors algorithm, showcase the diverse approaches employed to detect and resolve deadlocks.

## Reference

- Nabeel Aslam, Ahsan Ali, Arfan Shahz (2023). Comparative Analysis of Deadlock Detection Algorithm based on Blockchain. *International journal of computing, communication and networking*, 12(4), pp.12–16. doi: <https://doi.org/10.30534/ijccn/2023/011242023>.
- Kate, V., Jaiswal, A. and Ambika Gehlot (2016). A survey on distributed deadlock and distributed algorithms to detect and resolve deadlock. doi: <https://doi.org/10.1109/cdan.2016.7570873>.
- Agarwal, R., Bensalem, S., Farchi, E., Havelund, K., Nir-Buchbinder, Y., Stoller, S.D., Ur, S. and Wang, L. (2010). Detection of deadlock potentials in multithreaded programs. *IBM Journal of Research and Development*, 54(5), pp.3:1–3:15. doi: <https://doi.org/10.1147/jrd.2010.2060276>.
- Nasser, M. and Seyed Mahdi Jameii (2017). Concurrency control methods in distributed database: A review and comparison. doi: <https://doi.org/10.1109/comptelix.2017.8003964>.
- Caicedo Acosta, L., Ospina Acosta, C.A., Gelvez García, N.Y. and Romero Villalobos, O.A. (2014). Design of a Mutual Exclusion and Deadlock Algorithm in PCBSD – FreeBSD. *International Journal of Interactive Multimedia and Artificial Intelligence*, 3(1), p.44. doi: <https://doi.org/10.9781/ijimai.2014.316>.
- Rgibi, A., Alarbish, A. and Oshah, A. (n.d.). Deadlock Detection in Concurrency System using Timed Petri Net. *International Journal of Advance Research, Ideas and Innovations in Technology Impact*, [online] 8. Available at: <https://www.ijariit.com/manuscripts/v8i1/V8I1-1182.pdf> [Accessed 29 Apr. 2024].
- Hassan, M.H., Darwish, S.M. and Elkaffas, S.M. (2022). An Efficient Deadlock Handling Model Based on Neutrosophic Logic: Case Study on Real Time Healthcare Database Systems. *IEEE Access*, [online] 10, pp.76607–76621. doi: <https://doi.org/10.1109/ACCESS.2022.3192414>.

- Haque, W., Fontaine, M. and Vezina, A. (2017). Adaptive Deadlock Detection and Resolution in Real-Time Distributed Environments. doi: <https://doi.org/10.1109/hpcc-smartcity-dss.2017.74>.
- Zarnay, M. and Fernando Tricas Garcia (2014). Enhancing banker's algorithm for avoiding deadlocks in systems with non-sequential processes. doi: <https://doi.org/10.1109/etfa.2014.7005149>.